Representation Learning for Partitioning Problems

Corinne L. Jones

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Zaid Harchaoui, Chair

Yen-Chi Chen

Kevin Jamieson

Program Authorized to Offer Degree:
Statistics

University of Washington

Abstract

Representation Learning for Partitioning Problems

Corinne L. Jones

Chair of the Supervisory Committee:
Associate Professor Zaid Harchaoui
Statistics

This dissertation addresses representation learning for partitioning problems. Clustering a set of data points and segmenting a time series of data points are two classical partitioning problems. Nonparametric methods such as kernel-based methods assume the knowledge of a mapping into a feature space. Their statistical performance can, however, be impeded if this mapping, usually called a feature representation, is improperly specified or simply unknown. As larger datasets become available we can contemplate the possibility of, jointly, learning a feature representation and predicting clustering or segmentation labels.

The feature representations we consider here take the form of a nonlinear mapping built as a composition of basic modules, a computational skeleton commonly referred to as a deep network. The parameters of each module are learned from data using automatic differentiation techniques and gradient-based optimization algorithms. The combination of these allows us to tackle partitioning problems using end-to-end learning with gradient-based training algorithms. As a consequence, the proposed methods can improve upon classical kernel-based methods in terms of statistical performance as more data is used. The first half of this dissertation demonstrates this approach for learning feature representations for image categorization and clustering using convolutional kernel-based methods. The second half develops methods that broaden the scope of multiple change-point estimation in time series of data points using a feature representation mapping built from a data-dependent kernel

mapping or a neural network.

Chapter 2 explores the relationship between convolutional neural networks and related kernel-based convolutional networks. We show how to transform a neural network to a kernel network, providing a detailed mathematical description of each component of a network. We explore this comparison both analytically and empirically with milestone convolutional network architectures for image categorization and highlight the similarities and the differences between these two families of methods. Along the way we propose a gradient-based optimization method for training both neural and kernel networks.

Chapter 3 switches to learning feature representations in the presence of unlabeled data. We propose a single objective function that transitions between the supervised and unsupervised settings depending on the ratio of labeled to unlabeled data, recovering discriminative clustering at one end and supervised classification at the other end. We put in perspective the proposed method in the broader area of nonparametric similarity-based clustering methods and motivate the proposed objective for end-to-end learning. We propose to perform the clustering assignment using an entropy-regularized optimal transport algorithm. A numerical evaluation on several datasets demonstrates the interest of the approach.

Inspired by Chapter 3, Chapter 4 pivots to learning feature representations for change-point estimation. Similarly to Chapter 3, we develop a single objective function that can handle any ratio of labeled to partially labeled to unlabeled sequences. We propose two methods for optimizing the objective, one based on non-smooth optimization and the other based on smooth optimization. The numerical evaluation on synthetic and real-world data demonstrates the benefits of learning the feature representations for multiple change-point estimation compared to using fixed, pre-defined feature representations.

Finally, Chapter 5 proposes a change-point estimation method for data consisting of sequences of point clouds. We connect the method to the concept of distances between probability distributions and show how to scale up the approach when there are thousands of

point clouds, each with thousands of points. This work is motivated by an oceanographic application in which flow cytometry point cloud data on phytoplankton is collected underway during research cruises. We illustrate the utility of the proposed method on a flow cytometry dataset and the potential to estimate the number of change points using auxiliary data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, thanks to my advisor Zaid for taking me on as your student. Throughout the past four years you succeeded in instilling in me your unbridled passion for research. Your constant feedback, combined with this passion, propelled me to become a much more capable researcher and produce this dissertation. I have also benefited greatly from learning your tricks of the trade and receiving your endless advice.

Thanks to all of my former teachers, professors, and mentors who helped me achieve my dream of going to graduate school. Thanks to Kala for taking me on as your undergraduate research assistant, teaching me what research is like, always believing in me, and being supportive even when I decided to change directions. Thanks also to Bobby for introducing me to machine learning research, showing me how to write research papers, and providing wide-ranging advice.

Thanks to my colleagues here at the University of Washington. Thanks to Maryclare, Rebecca, Sam, and Wen Wei for being extremely welcoming when I came to UW and imparting your wisdom. Thanks to Alec Z., Amrit, Anne, Chris A., Eric, Hoiyi, Kyle, Luca, Wayne, and Wesley for our discussions about coursework and for all of the game nights the first two years. Thanks to Anna, Erik, and Jay for keeping me sane by playing tennis and/or soccer with me. Thanks to all of Zaid's current and former students: to Alec G.-T. for providing feedback on my work and answering miscellaneous questions; to Chris X. for our discussions and for giving me feedback on my slides; to John for giving me advice, teaching me a lot about GPU computing, and providing feedback on my work; to Krishna for helping me troubleshoot GPU problems and providing feedback on my papers and presentations; and to Lang, Meyer, Romain, and Yassine for your help addressing miscellaneous problems. Thanks too to Sean

for feedback and advice. Finally, thanks to Daphne, Ema, Hannah, Laina, and Mariana for being there at the times when I needed it the most.

Thanks to all of the collaborators I've worked with. Thanks especially to Vincent for being a great mentor, answering all of my questions, and always being helpful. Thanks also to François, Ginger, and Sophie for introducing me to the world of oceanography and patiently answering my naïve questions.

Thanks to all of the UW professors, instructors, and staff I have had the pleasure of interacting with. Thanks to Kevin, Maya, Werner, and Yen-Chi for serving on my committees. Thanks to Caren for showing me how to produce high-quality teaching materials and for taking me on as your RA during my first summer here. Thanks to Daniela for standing up for me. Thanks to Marina for your feedback and advice. Thanks to Mathias for convincing me to come to UW and for playing tennis with me. Thanks to Paul for teaching me how to be a great statistical consultant. Thanks to Thomas for being supportive. Thanks to all of the staff, including Ellen, Kristine, Tracy, and Vickie, for all of your help in answering various questions.

Finally, thanks to my friends and family. Thanks to Carrie, Clara, and Sherry for being great friends, and to Eric, Justine, and Tarun for being great friends and housemates. Thanks to Gwen and Rick for being so supportive. Above all, thanks to my parents and sister, Therese. I could not have gotten this far without you.

DEDICATION

To my family.

Chapter 1

INTRODUCTION

As George Box once wrote, "All models are wrong but some are useful" (Box, 1979). The first half of this dissertation focuses on deep learning models for classification and clustering. Over the past decade deep learning models have proven extremely useful in many domain applications. For example, convolutional network models are being used to help diagnose lung cancer (Ardila et al., 2019), recurrent network models are being used to recognize speech (Graves et al., 2013), and transformer network models are being used to translate text from one language to another (Vaswani et al., 2017). However, just as Box said, these deep learning models are still wrong. In fact, there are several important issues that concern many deep learning models. For example, seemingly minor changes to inputs can lead to drastically different predictions (Szegedy et al., 2014). Even more worrisome, without proper development, models are often biased and lack robustness to harmful data (Plaugic, 2017; Risley, 2016). Some of these problems with deep learning (and machine learning more broadly) stem from the fact that the model engineering facet of research has progressed faster than the theory. As a result, deep learning can presently be viewed as a "dense jungle", with an enormous number of models. Oftentimes the models themselves and the connections between the models are poorly understood.

In the first half of this dissertation we begin to make sense of a portion of this jungle and propose deep learning approaches that are more principled than what presently exists. Specifically, in Chapter 2 we explore the relationship between convolutional neural networks and kernels. We transform convolutional neural networks to kernel networks and examine the performance of each. Then, in Chapter 3, we propose a deep learning method for clustering when any fraction of the data may be labeled.

The second half of this dissertation focuses on models for change-point estimation. The goal of change-point estimation is to locate changes in distribution within a sequence of observations. As we will see in Section 1.2, there also exist areas of change-point estimation that remain unexplored. Chapter 4, inspired by Chapter 3, proposes a method for learning feature representations for change-point estimation when any fraction of the training sequences may be labeled. Chapter 5 develops a change-point estimation method for sequences of point clouds and applies it to oceanographic data.

In the next section we review deep networks and kernels, outline a number of challenges in deep learning, and describe where the next two chapters of this dissertation fit in. We then proceed in Section 1.2 to provide an overview of the landscape of change-point methods and describe where Chapters 4 and 5 lie.

1.1 Deep Learning

Often data comes in a form that is not readily amenable to statistical analysis. For decades researchers have therefore applied hand-crafted transformations to data, resulting in, for example, SIFT features for images, MFCC coefficients for speech, and parse trees for natural language (Lowe, 1999; Davis and Mermelstein, 1980; Manning and Schütze, 2001). The more recent successes of deep learning have demonstrated that it is typically more effective to automatically learn how to transform the original representations of data into more useful representations for analysis.

Traditional deep learning workflows have three components that need to be specified: the model, the loss function, and the optimization procedure. The model is a function that transforms its inputs in a nonlinear manner. Depending on the model, it can have thousands or even millions of parameters. The loss function provides a means of assessing how similar the output of the model is to the ground truth. Finally, given (large) sets of labeled inputs $x_1, x_2, \ldots, x_n$ and target outputs $y_1, y_2, \ldots, y_n$, the optimizer provides a means of learning what the parameters should be in order for the outputs of the model to match the target outputs. The optimization is typically done "end-to-end", which entails differentiating the

loss function with respect to the model parameters and using the resultant gradients to update the model parameters. The gradients are typically computed using a method called "backpropagation". The goal is generally to be able to learn a model that generalizes well to previously unseen data.

In the remainder of this section we first review two types of neural network models. We then outline four open problems in deep learning. Afterward, we review kernel methods, which are used throughout this dissertation. Finally, we discuss how the contents of this dissertation relate to the outlined open problems.

1.1.1 Neural networks

In this dissertation we examine two types of neural network models: multi-layer perceptrons (MLPs) and convolutional neural networks (ConvNets). An MLP takes as input an observation $x \in \mathbb{R}^{f_0}$. Here f_0 is the number of input features (i.e., variables). This input gets passed sequentially through a series of L layers. The input to each layer ℓ gets pre-multiplied by a matrix of learnable weights $W_\ell \in \mathbb{R}^{f_\ell \times f_{\ell-1}}$ and then a learnable bias vector $b_\ell \in \mathbb{R}^{f_\ell}$ is added. Finally, a nonlinear function $a_\ell : \mathbb{R} \to \mathbb{R}$ (called a nonlinearity or activation function) gets applied element-wise. Mathematically, all of this can be expressed recursively as

$$
\begin{aligned}
F_0^{MLP}(x) &= x \\
F_\ell^{MLP}(x) &= a_\ell(W_\ell F_{\ell-1}^{MLP}(x) + b_\ell) \ , \quad \ell = 1, \ldots, L-1 \\
F_L^{MLP}(x) &= W_L F_{L-1}^{MLP}(x) + b_L \ ,
\end{aligned}
$$

where a is understood to be applied element-wise. The vector F_L^{MLP} is called the (output) feature representation for x. Typically the nonlinearity a_ℓ is taken to be the same for all layers ℓ.

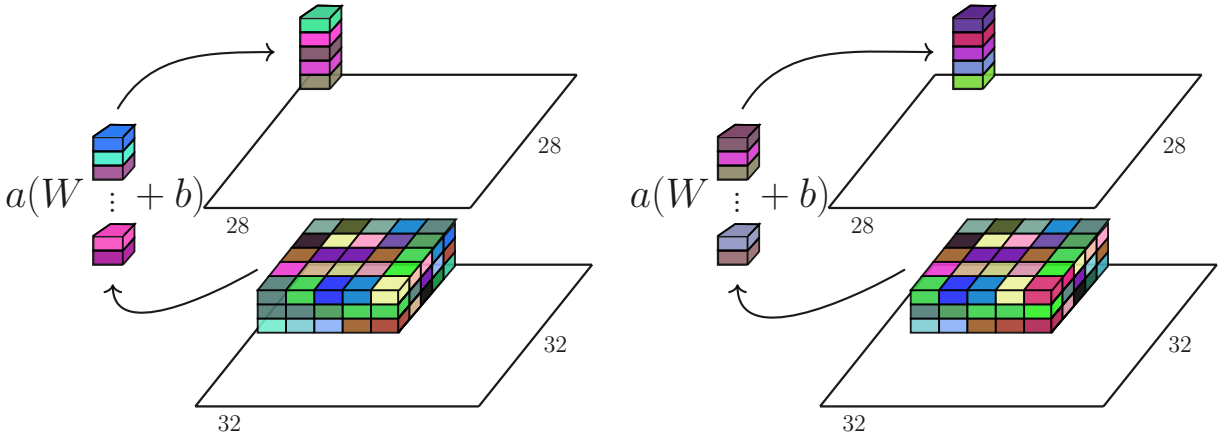
ConvNets are extensions of MLPs whose inputs are generally multi-dimensional. Basic ConvNets have three main components: convolutions, nonlinearities, and pooling. Let $x \in \mathbb{R}^{f_0 \times h_0 \times w_0}$ be an input to a ConvNet. Here h_0 and w_0 are the spatial dimensions and f_0

is the number of input channels (e.g., if x is a color image in RGB format, then $f_0 = 3$). For expository purposes we will collapse the spatial dimensions into one dimension and write $x_0 \in \mathbb{R}^{f_0 \times p_0}$ where $p_0 = h_0 w_0$. Computationally, a convolution first extracts contiguous patches of some size s_1 from the inputs, resulting in $E_1(x_0) \in \mathbb{R}^{s_1 \times p_1}$. Afterward, this matrix of patches is pre-multiplied by a weight matrix $W_1 \in \mathbb{R}^{f_1 \times s_1}$. Finally, a bias vector $b_1 \in \mathbb{R}^{f_1}$ is added to each column of the result, yielding $W_1 E_1(x_0) + b_1 \mathbb{1}^T \in \mathbb{R}^{f_1 \times p_1}$, where $\mathbb{1}$ is a vector of 1's of dimension p_1. The nonlinearity in a ConvNet gets applied element-wise to the result of the convolution. A common nonlinearity is the ReLU, defined as $a(z) = \max\{z, 0\}$. Finally, a pooling function c_ℓ may be applied. Two common types of pooling are max pooling and average pooling. These functions either take the maximum (for max pooling) or average (for average pooling) of elements in contiguous patches. The functions are applied channel-wise (i.e., row-wise) on the output from the nonlinearity.

A set of the aforementioned three components of a ConvNet comprise a "layer", and a ConvNet consists of multiple layers. We write the output of the ConvNet as $F_L^{CN}(x)$, which may be defined recursively as

$$
\begin{aligned}
F_0^{CN}(x) &= x_0 \\
F_\ell^{CN}(x) &= c_\ell(a_\ell(W_\ell E_\ell(F_{\ell-1}^{CN}(x)) + b_\ell \mathbb{1}^T)) \ , \quad \ell = 1, \ldots, L-1 \\
F_L^{CN}(x) &= W_L F_{L-1}^{CN}(x) + b_L \mathbb{1}^T \ .
\end{aligned}
$$

Figure 1.1 illustrates 5×5 spatial convolutions followed by a nonlinearity in a ConvNet (without collapsing the spatial dimensions). The input to the layer has spatial dimensions 32×32 and depth 3. A 5×5 patch (with depth 3) is first extracted from the image. This patch is then vectorized, with length 75. This vectorized patch is pre-multiplied by a weight matrix $W \in \mathbb{R}^{5 \times 75}$ and afterward a bias vector $b \in \mathbb{R}^5$ is added. Finally, the nonlinearity a is applied. This is then repeated for the next patch, which is shifted by one unit to the right. The procedure continues until it has been applied to all contiguous 5×5 patches in the input. The output is therefore of size $28 \times 28 \times 5$.



(a) Transformation of the first patch (b) Transformation of the second patch

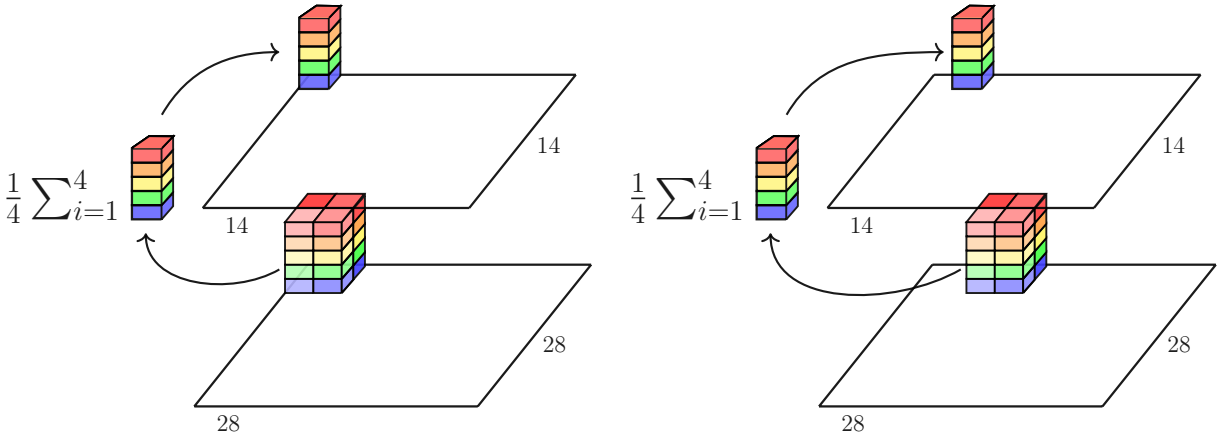
Figure 1.1: Illustration of the convolution followed by a nonlinearity in a ConvNet.

Figure 1.2 illustrates 2×2 spatial pooling with stride 2. First a patch of size 2×2 (with depth 5) is extracted. Then each horizontal slice is averaged. The result is a $1 \times 1 \times 5$ vector. Afterward, this is performed for the next patch, which is a stride of two away from the first patch. This continues until the operation has been performed on all contiguous 2×2 patches whose L_1 distances from the initial patch are a multiple of two.

Finally, we introduce some additional terminology commonly used when referring to ConvNets. First, each row of W is often called a "filter". Moreover, each row of F_ℓ is called a "feature map" or a "channel". Finally, the "receptive field" of an entry of F_ℓ consists of the entries in the input F_0 that were used to produce that entry of F_ℓ.

1.1.2 Open questions in deep learning

The resurgence of neural networks to the forefront of machine learning research has brought renewed interest in their modeling and optimization. We now proceed to highlight four major questions in deep learning, and briefly review progress in these areas.



(a) Pooling of the first patch (b) Pooling of the second patch

Figure 1.2: Illustration of average pooling in a ConvNet.

Is there a principled way to design networks? From the description of ConvNets above we can see that the number of hyperparameters that need to be specified when building even a basic ConvNet is overwhelming. The hyperparameters include the number of layers, the number of filters per layer, the stride of each convolution and pooling operation at each layer, the channel connectivity pattern at each layer, the type of pooling at each layer, the spatial dimensions of the filters at each convolutional layer, the size of the pooling region at each pooling layer, the type of nonlinearity at each convolutional layer, and the number and ordering of the pooling and convolutional layers. There have been papers suggesting that pooling may not be necessary (Springenberg et al., 2015) and that certain activation functions are superior to others (Ramachandran et al., 2018). Even removing these two hyperparameters leaves an enormous search space.

Within the past several years various methods for searching the hyperparameter space have been proposed. Saxena and Verbeek (2016) proposed an elegant approach that embeds most of the hyperparameters in a neural "fabric" that is learned. In this approach only the number of layers and the number of filters per layer are left as hyperparameters. The downside to this approach is that it is unclear that an optimization algorithm would in fact recover a

high-quality fabric, and the existence of this flaw is backed up by the empirical results. Other authors propose to perform neural architecture search. The seminal work in this area used reinforcement learning and trained over 12,000 architectures (over 1350 GPU-days) in order to find an architecture for the image dataset CIFAR-10 (Zoph and Le, 2017). Several recent papers present more economical approaches based on continuous relaxations that take only a day or two to train (Liu et al., 2019; Xie et al., 2019). However, tackling a huge search space for all of the hyperparameters remains computationally intractable.

What networks generalize well? The generalization abilities of deep networks are still poorly understood. As shown by Zhang et al. (2017), state-of-the-art ConvNets can fit data with random labels nearly perfectly. They can also nearly perfectly fit data with unstructured, random noise as inputs. Since these observations are unaffected by the addition of an explicit regularization penalty, neither the model family nor the regularization techniques seem to explain this behavior. As Belkin et al. (2018) noted, similar results can also be obtained with kernel methods. However, recent work by Schmidt-Hieber (2020) showed that deep neural networks with ReLU activation functions can outperform classical nonparametric statistical estimators, in that they are able to adapt to the underlying compositional structure of the regression function when such a compositional structure exists.

One approach to understanding the generalization of networks is to prove generalization bounds. Many generalization bounds have been proposed, but few seem to match the empirical behavior observed in practice in terms of generalization performance (Jiang et al., 2020). One approach to closing this gap between the theoretical analysis and the empirical behavior is to optimize the parameters in a simplified PAC-Bayes bound (Dziugaite and Roy, 2017). However, the resultant bounds are still loose and much room remains for improvement.

How can deep networks be made interpretable? Due to the highly nonlinear nature of deep networks, it is difficult to interpret why they make the decisions they do. A number of methods have been proposed for examining ConvNets after they have been trained. To

find the prototypical image for a given class one can optimize over the space of images with the network fixed to find the image that maximizes the score for that class. In addition, to understand the parts of a given image that most influence this score one can differentiate with respect to the image (Simonyan et al., 2014). Moreover, to understand what parts of an image most influence a specific element in a feature map, one can train a network to reconstruct the input (Zeiler and Fergus, 2014).

More recently there has been work on building interpretability into ConvNets. In particular, Chen et al. (2019) proposed adding an additional layer before the output layer of a network called a "prototype layer" that is applied to feature representations. Each prototype is a vector, and there are a given number of prototypes per image class. At the prototype layer the similarity of each prototype to each 1×1 patch in the feature representation is computed. The output of the layer is the maximum similarity over all patches for each prototype. During training the prototypes get projected to the nearest patch in a feature representation from the prototype's assigned class. In order to understand what parts of an image lead to a large similarity score with a prototype, one can upsample the similarities between the prototype and the patches in the image's feature representation and then locate the portion of the original image corresponding to the largest similarities.

How should unlabeled data be used in classification tasks? In addition to designing a network, one must also choose an objective function. In the supervised setting, in which every input in the training set comes with a label, one can perform, e.g., multinomial logistic regression on the feature representations output by the network. However, in the semi-supervised and unsupervised settings, where some or none of the inputs are labeled, it is not so straightforward. Optimizing simultaneously over the network parameters and unknown labels can lead to trivial solutions in which all inputs are assigned the same label (Bach and Harchaoui, 2007). Researchers have developed an assortment of creative approaches that avoid this problem, including learning feature representations on auxiliary tasks (e.g., Doersch et al., 2015; Hyvärinen and Morioka, 2016) and reassigning empty clusters (e.g., Caron et al.,

2018).

1.1.3 Kernel methods

An alternative approach to generating feature representations, which has historically been thought of as distinct from neural network-based approaches, is based on kernels. Informally, a kernel can be thought of as a pairwise similarity function. More formally, given a non-empty set $\mathcal{X}$, a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a reproducing kernel if there exists a Hilbert space $\mathcal{H}$ (i.e., a finite- or infinite-dimensional space equipped with a dot product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$) and a mapping $\phi : \mathcal{X} \to \mathcal{H}$ such that for all $x, x' \in \mathcal{X}$,

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$$

(Steinwart and Christmann, 2008). One of the most popular kernels is the Gaussian RBF kernel, given by $k(x, x') = \exp(-\|x - x'\|^2/(2\sigma^2))$.

Kernel-based methods replace an input x by $\phi(x)$, thereby mapping x to a feature representation for x. While working with ϕ itself is generally computationally intractable, many methods only rely on dot products of features, making learning feasible. As a concrete example, consider inputs $x_1, x_2, \ldots, x_n \in \mathcal{X}$ and labels $y_1, y_2, \ldots, y_n \in \{-1, +1\}$. Then for a given kernel k with canonical feature map ϕ, substituting $\phi(x_i)$ for each x_i in the objective of an ℓ_2^2-regularized linear SVM with the hinge loss yields the problem

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle f, \phi(x_i) \rangle_{\mathcal{H}}\} + \lambda \|f\|_{\mathcal{H}}^2 .$$

By the representer theorem (Kimeldorf and Wahba, 1971), we know the minimizer is of the form $\sum_{i=1}^{n} \beta_i \phi(x_i)$ where $\beta_i \in \mathbb{R}$ for all i. Therefore, defining the Gram matrix $K = [k(x_i, x_j)]_{i,j=1}^{n}$, the problem may be written as

$$\min_{\beta \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} \max\left\{0, 1 - y_i \beta^T K_{\cdot,i}\right\} + \lambda \beta^T K \beta , \tag{1.1}$$

i.e., based only on inner products. An advantage of kernel methods is that they can work on any kind of data, including images, graphs, and text, as long as a kernel can be defined.

When the number of inputs n is large, computing K can be computationally intractable. In this case one can approximate the kernel using an approximate feature map $\psi : \mathcal{X} \to \mathbb{R}^d$, such that for all $x, x' \in \mathcal{X}$, $k(x, x') \approx \langle \psi(x), \psi(x') \rangle_{\mathbb{R}^d}$. There are numerous ways to do this, two of which we discuss in Chapter 2. Throughout this dissertation we approximate and compose kernels in order to learn feature representations.

1.1.4 Contributions of this dissertation

The next two chapters of this dissertation contribute to the understanding of (convolutional) neural networks. Chapter 2 demonstrates how convolutional neural networks can be transformed into kernel counterparts, called convolutional kernel networks (CKNs). CKNs consist of compositions of kernels, where each kernel is approximated using learned parameters. The kernel viewpoint provides interesting insights. The weights and nonlinearities are used to approximate kernels. Furthermore, the number of filters in each weight matrix determines the quality of the approximation. This understanding paves the way for the potential to design networks based on desired attributes, including the desired approximation quality of the overall kernel. Moreover, if certain invariances are desired, these can be incorporated via the choice of kernels. Since each layer can be viewed as generating a feature map to approximate a kernel, it is easier to interpret the resulting learned network. Indeed, determining which (patches of) inputs are similar to which other (patches of) inputs as Chen et al. (2019) did amounts in principle here to simply computing inner products; no additional layers or optimization is necessary.

In Chapter 2 we also provide a systematic experimental comparison of ConvNets and their CKN counterparts. This study provides empirical evidence for when kernel networks can and cannot perform as well as ConvNets. The existence of such an empirical assessment can help inform future theoretical work on understanding the performance and generalization of CKNs. In this chapter we also study the smoothness of the CKNs and propose a new

algorithm for training networks based on a variable elimination strategy reminiscent of the one used in profile likelihood (Barndorff-Nielsen and Cox, 1994).

Chapter 3 switches to learning for classification tasks in the presence of labeled and unlabeled data. Most of the popular methods for doing this are either entirely unsupervised (e.g., Caron et al., 2018) or do not learn feature representations (e.g., Bach and Harchaoui, 2007; Xu et al., 2009). In this chapter we propose a single objective function that unifies unsupervised and supervised learning. In the case where there are no labels, the algorithm performs clustering. In the case where all of the labels exist, it performs classification. In between, when some labels exist, the algorithm is a combination of clustering and classification. We develop a specialized algorithm for optimizing this objective and demonstrate in the experiments that using the unlabeled data often improves the accuracy of the learned classifier.

1.2 Change-Point Analysis

The second half of this dissertation relates to change-point methods. Change-point methods are used to locate and/or test for changes in distribution in a sequence of observations. Figure 1.3 illustrates change points in a 1-dimensional sequence of observations. In the figure there are three change points (changes in mean), at indices $t = 15$, $t = 45$ and $t = 65$, and these change points are highlighted with vertical red lines.

One of the most basic change-point estimation methods is the normal single-change-in-mean model. Suppose there exists a sequence of observations $x_1, x_2, \ldots, x_T \in \mathbb{R}^d$ for some d. Assume that x_t is an independent realization of a random variable drawn from $N(\mu^t, \sigma^2 \mathrm{I}_d)$ for all t where σ is fixed but unknown and there exists only one index t at which $\mu^{t-1} \neq \mu^t$. This index, which we will denote t_1, divides the sequence into two segments, one from observations 1 through $t_1 - 1$, and one from observations t_1 through T. Denote the mean of the first segment of observations by μ_0 and that of the second segment by μ_1. The parameters t_1, μ_0, and μ_1 can be estimated via maximum likelihood estimation. Concretely, defining $t_0 \coloneqq 1$

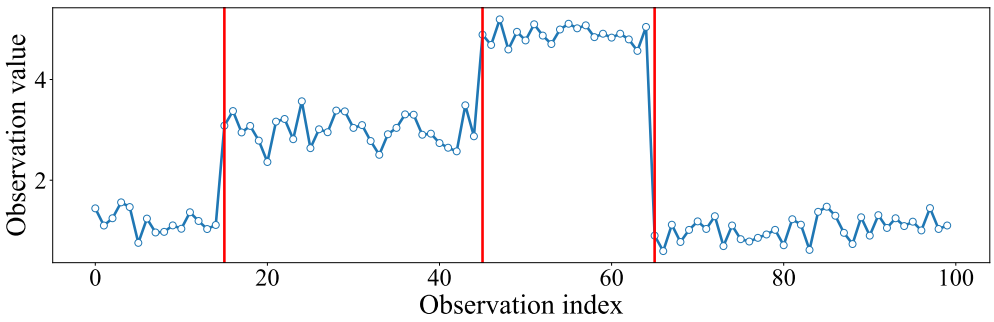


Figure 1.3: Example of a one-dimensional sequence of Gaussian observations with changes in mean at indices $t = 15$, $t = 45$, and $t = 65$.

and $t_2 \coloneqq T + 1$, the maximum likelihood problem may be written as

$$\max_{t_1,\mu_0,\mu_1} \prod_{j=0}^{1} \prod_{t=t_j}^{t_{j+1}-1} \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{1}{2\sigma^2}\|x_t - \mu_j\|_2^2\right) .$$

It is straightforward to show that the maximum likelihood estimate of μ_j is the segment mean $\hat{\mu}_j = 1/(t_{j+1} - t_j) \sum_{t=t_j}^{t_{j+1}-1} x_t$ for $j = 0, 1$. Hence, after simplification, the problem of estimating the change-point location boils down to

$$\min_{t_1} \quad \frac{1}{T} \sum_{j=0}^{1} \sum_{t=t_j}^{t_{j+1}-1} \|x_t - \hat{\mu}_j\|_2^2 . \tag{1.2}$$

This problem can be solved via brute force search over all possible values of t_1.

The problem of maximizing the likelihood in this setting is closely related to maximizing a likelihood ratio. Suppose μ_0 is known. Then finding the index t_1 to minimize the above criterion is the same as finding t_1 to minimize $\sum_{t=t_0}^{t_1-1} \|x_t - \mu_0\|_2^2 + \sum_{t=t_1}^{t_2-1} \|x_t - \hat{\mu}_1\|_2^2 - (\sum_{t=t_0}^{t_1-1} \|x_t - \mu_0\|_2^2 + \sum_{t=t_1}^{t_2-1} \|x_t - \mu_0\|_2^2)$. The minimizer is the same as the maximizer of

$$\max_{t_1} \quad \sum_{t=t_1}^{t_2-1} \left\{\|x_t - \mu_0\|_2^2 - \|x_t - \hat{\mu}_1\|_2^2\right\} .$$

This objective is precisely a scaled version of the log of the generalized likelihood ratio statistic of the hypothesis $H_0 : \mu^t = \mu_0$ for $t_1 \leq t \leq T$ vs. $H_1 : \mu^t \neq \mu_0$ for $t_1 \leq t \leq T$.

The objective (1.3) can be readily extended in the case where there is more than one change point. The resultant normal change-in-mean objective for the case of m change points is given by

$$\min_{t_1,\ldots,t_m} \quad \frac{1}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \|x_t - \hat{\mu}_j\|_2^2 \,. \tag{1.3}$$

This objective can be optimized efficiently via dynamic programming and has computational complexity $O(mT^2)$ (Fisher, 1958; Bellman, 1961; Kay, 1993).

The change-point literature is rather vast, and methods developed in prior work can be categorized via a lot of different attributes. In the remainder of this section we first elaborate on many of these attributes, showing how the normal change-in-mean method can be adapted to obtain methods in other categories. We then discuss where the contributions of this dissertation lie.

1.2.1 Classification of change-point methods

In this subsection we delineate categories of change-point methods and demonstrate how the normal change-in-mean model can be modified to have different characteristics. Throughout the discussion we will assume that there are multiple change points and that the locations of the changes are unknown. Table 1.1 provides a diverse set of examples of methods that can be used to identify multiple change points in sequences of observations.

Bayesian vs. frequentist. The normal change-in-mean approach is frequentist. We can instead take a Bayesian approach by putting priors $p(\mu_j)$ on the model parameters for $j = 0, \ldots, m$, a prior $p(m)$ on the number of change points, and a prior $p(t_1, \ldots, t_m|m)$ on the locations of the change points given the number of change points. Defining $x = \{x_1, \ldots, x_T\}$ and $\mu = \{\mu_0, \ldots, \mu_m\}$, and observing that $p(m, t_1, \ldots, t_m|x) = \int p(m, t_1, \ldots, t_m, \mu|x)d\mu \propto$

Method	Bayesian?	Estimate states?	Generic data types?	Hypothesis testing?	Learn features?	Nonparametric?	Online?
Normal change-in-mean (Quandt, 1958)							
Bayesian approach (Fearnhead, 2006)	✓						
HVGH (Nagano et al., 2019)	✓	✓			✓	✓	
Sticky HDP-HMM (Fox et al., 2011)	✓	✓				✓	
Online Bayesian approach (Fearnhead and Liu, 2007)	✓						✓
HMM (Baum and Petrie, 1966; Cappé, 2011)		✓					B
HS embeddings of conditional dist. (Song et al., 2009)		✓	✓			✓	✓
KCPE (Harchaoui and Cappé, 2007)			✓			✓	
Maximum KFDR (Harchaoui et al., 2008)			✓	✓		✓	B
XSCPE (Jones and Harchaoui, 2020)			✓		✓		
E-divisive (Matteson and James, 2014)				✓		✓	
GLR (Lai and Xing, 2010)				✓			✓
Nonparametric maximum likelihood (Zou et al., 2014)						✓	

Table 1.1: Examples of change-point estimation algorithms with various characteristics. The entry "B" in the "Online?" column denotes "both", i.e., it could be used as an online or an offline algorithm.

$\int p(x, \mu | t_1, \ldots, t_m) p(t_1, \ldots, t_m | m) p(m) d\mu$, we can compute the maximum a posteriori (MAP) estimates of the change points by solving

$$\max_{m, t_1, \ldots, t_m} \int \prod_{j=0}^{m} \left\{ \left[\prod_{t=t_j}^{t_{j+1}-1} p(x_t | \mu_j) \right] p(\mu_j) \right\} p(t_1, \ldots, t_m | m) p(m) d\mu \, . \tag{1.4}$$

In practice, depending on the choice of priors, this problem can be intractable. In that case, we could instead consider sampling from the posterior, as done by Fearnhead (2006). In the case where the change-point locations are uniformly distributed conditional on the number of change points, exact inference can be possible. In this case the computational complexity is $O(mT^2)$.

Correlated vs. uncorrelated data within segments. While the normal change-in-mean model assumes that the data within each segment is independent and identically distributed (i.i.d.), this is not always the case in practice. Instead, the data may, for example, follow an autoregressive process. In this case, we can change the model accordingly (Bai, 2000):

$$x_t = \mu_j + \sum_{r=1}^{p} A_{jr} x_{t-r} + \Sigma_j^{1/2} e_t \quad \text{for all } t = t_j, \ldots, t_{j+1} - 1 \ ,$$

where p is the order of the model, $\mu_j \in \mathbb{R}^d$ for all j are the constant terms, $A_{j,p} \in \mathbb{R}^{d \times d}$ for all j, p are the AR coefficients, $\Sigma_j^{1/2} \in \mathbb{R}^{d \times d}$ are the covariance matrices, and e_t is an error vector drawn from $N(0, \mathrm{I}_d)$. Bai (2000) proposes estimating the change points via the quasi-maximum likelihood objective

$$\min_{\substack{t_1,\ldots,t_m,\\ \mu,A,\Sigma}} \frac{1}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \left\{ \left(x_t - \mu_j - \sum_{r=1}^{p} A_{jr} x_{t-r} \right)^T \Sigma_j^{-1} \left(x_t - \mu_j - \sum_{r=1}^{p} A_{jr} x_{t-r} \right) + \log |\Sigma_j| \right\} \ ,$$

where $\mu = \{\mu_0, \ldots, \mu_m\}$, $A = \{A_{j,r}, r = 1, \ldots, p, j = 0, \ldots, m\}$, and $\Sigma = \{\Sigma_0, \ldots, \Sigma_m\}$. The computational complexity is $O(mT^2)$.

Estimates vs. does not estimate states and transitions. In a time series of observations with change points one may have underlying states that determine the distributions of the observations. The normal change-in-mean model does not estimate the states, but we could modify it to do so. Consider a hidden Markov model with states $1, 2, \ldots, S$, where the number of states S is known. Let z_t be the value of the hidden state at time t and assume that the distribution of z_t given the past and future states is $p(z_t | z_1, \ldots, z_{t-1}, z_{t+1}, \ldots, z_T) = p(z_t | z_{t-1})$ for all t, i.e., it satisfies the Markov property. Then defining p to be the $S \times S$ matrix of state transition probabilities and π_0 to be the distribution of the initial states z_0 at time 0,

we obtain the (scaled) complete-data negative log-likelihood problem

$$\min_{z_1,\ldots,z_T,\mu_1,\ldots,\mu_S,\sigma,P,\pi_0} \frac{1}{T}\sum_{s=1}^{S}\sum_{t:z_t=s}\|x_t-\mu_s\|_2^2 + 2d\sigma^2\log(\sigma) - \frac{2\sigma^2}{T}\log(\pi_0) - \frac{2\sigma^2}{T}\sum_{t=1}^{T}\log p(z_t|z_{t-1}).$$

The optimization can be performed via alternating minimization, with the minimization over the hidden state sequence performed via dynamic programming using the Viterbi algorithm (Viterbi, 1967; Juang and Rabiner, 1990). Alternatively, we can minimize the incomplete-data negative log-likelihood using the EM algorithm (Baum et al., 1970; Bilmes, 1997). Afterward, we can obtain the estimated change points from the estimated hidden state sequence. The computational complexity of each iteration of either approach is $O(S^2T)$.

Exact vs. approximate. If the length T of a sequence is large, solving the normal change-in-mean problem (1.3) can be computationally expensive or even intractable. Note that we can reframe Problem (1.3) as $\min_{\mu_0,\ldots,\mu_m}\sum_{t=1}^{T}\|x_t-\mu_j\|_2^2/T$ subject to $\sum_{t=1}^{T-1}\mathbb{1}\{\mu_t \neq \mu_{t+1}\} = m$. To obtain an approximate solution to this problem more quickly, Harchaoui and Lévy-Leduc (2010) propose replacing the constraint by the sparsity-inducing ℓ_1 constraint $\sum_{t=1}^{T-1}\|\mu_{t+1}-\mu_t\|_1 \leq s$ for some constant s:

$$\begin{aligned} \min_{\mu_0,\ldots,\mu_m} \quad & \frac{1}{T}\sum_{t=1}^{T}\|x_t-\mu_j\|_2^2 \\ \text{subject to} \quad & \sum_{t=1}^{T-1}\|\mu_{t+1}-\mu_t\|_1 \leq s\ . \end{aligned}$$

They show how to solve this revised problem in the one-dimensional case via a change of variables, and suggest using it to find the largest $\bar{m} > m$ non-zero changes in the estimated mean vectors. They then run a modified version of the dynamic programming algorithm used to solve (1.3) in order to reduce the number of potential change points down to m. The computational complexity of finding the first $\bar{m}$ changes in this manner is $O(\bar{m}^3 + \bar{m}^2T)$ and the modified version of the dynamic programming algorithm is $O(\bar{m}^3)$. Hence, the

computational complexity of the overall procedure is linear in the length of the sequence rather than quadratic.

Fixed vs. estimated number of change points. The normal change-in-mean model assumes that the number of change points is fixed to some value m. In practice, we often do not know the true number of change points. One way to estimate the value of m is via adding a penalty term to the objective function and optimizing over m as well:

$$\min_{m=0,1,\ldots,m_{\max}} \min_{t_1,\ldots,t_m} \quad \frac{1}{T}\sum_{j=0}^{m}\sum_{t=t_j}^{t_{j+1}-1} \|x_t - \hat{\mu}_j\|_2^2 + \frac{m+1}{T}\sigma^2\left(c_1 \log\left(\frac{T}{m+1}\right) + c_2\right) ,$$

where $c_1, c_2 \in \mathbb{R}$ are positive constants and $m_{\max}$ is the maximum allowable number of change points (Lebarbier, 2005). With this form of a penalty it can be shown that the risk of the resultant estimator is close to that of the risk of the estimator with the ideal number of change points. Lebarbier (2005) suggests using $c_1 = 2$ and $c_2 = 5$. The value of σ^2 can be estimated using the slope heuristic (Baudry et al., 2012). Given values of c_1, c_2, and σ^2, the computational complexity of computing the solution to this problem is $O(m_{\max}T^2)$.

Model-based vs. model-free. Rather than considering a parametric model as in the normal change-in-mean case, we could use a model-free approach. For example, consider computing the quantity

$$\max_{t_1} \frac{1}{T}\sum_{s=1}^{t_1-1}\sum_{t=t_1}^{T} \|x_s - x_t\|_2^\alpha - \frac{T-t_1+1}{(t_1-2)T}\sum_{1\le s<t<t_1} \|x_s - x_t\|_2^\alpha - \frac{t_1-1}{(T-t_1)T}\sum_{t_1\le s<t\le T} \|x_s - x_t\|_2^\alpha ,$$

where the optimal t_1 represents an estimated change-point location. Assuming the observations before the change point are i.i.d. and similarly for the observations at and after the change point, then as long as $E[|X_t|^\alpha] < \infty$ for all t, this statistic scaled by $T/((t_1-1)(T-t_1+1))$ converges almost surely to its expectation as the number of samples in each segment goes to infinity. Moreover, under the null hypothesis of no change, the distribution of the statistic

will converge to a non-degenerate random variable and under the alternative hypothesis it will diverge (Matteson and James, 2014). To locate all changes, Matteson and James (2014) suggest performing binary segmentation and testing for the significance of each potential change point via a permutation test. In binary segmentation the statistic is first computed on the entire sequence. If the change point is determined to be significant, then the sequence is segmented into two parts. The procedure is repeated on each of the two parts, and continues to segment the sequence until no further potential change point is significant. If there are m change points then the computational complexity is $O(mT)$.

Online vs. offline. Instead of performing change-point estimation offline, i.e., examining the entire sequence at once, we could instead perform it online, examining observations sequentially. In the case where neither the mean μ_0 before a potential change point s nor the mean μ_1 after the change point is known, the extension of the (scaled) generalized likelihood ratio on observations at times $t = t_{j-1}, \ldots, t'$ is given by

$$\mathrm{GLR}_{t_{j-1}:s:t'} = \min_{\mu_0} \sum_{t=t_{j-1}}^{t'} \|x_t - \mu_0\|_2^2 - \min_{\mu_0,\mu_1} \left(\sum_{t=t_{j-1}}^{s-1} \|x_t - \mu_0\|_2^2 + \sum_{t=s}^{t'} \|x_t - \mu_1\|_2^2 \right) .$$

An alert for a change point is raised if

$$\max_{s \in (t_{j-1}, t']} \mathrm{GLR}_{t_{j-1}:s:t'} \geq \sigma^2 c$$

for some parameter c. The estimated change point is the argmax of this expression, which we denote by t_j. After such a change point is detected the sequence is reset to start at t_j. Lai and Xing (2010) study this problem in the case where the observations come from a multivariate exponential family. They suggest setting the parameter c based on a desired false alarm average run length. The computational complexity is $O(mT)$, where m is the number of detected changes.

Parametric vs. nonparametric. Rather than assuming that the observations are normally distributed as in the normal change-in-mean case, we can instead take a nonparametric approach. Assume the observations within each segment j are univariate and real-valued and have a cumulative distribution function (CDF) $F_j : \mathbb{R} \to [0, 1]$. The idea behind the nonparametric maximum likelihood approach of Zou et al. (2014) is to maximize the nonparametric likelihood of $F_j(u)$ across all $u \in \mathbb{R}$. Concretely, consider realizations $x_1, \ldots, x_s$ of random variables drawn from a distribution with CDF F_0. We can classify each observation by whether or not it is less than a given value u. The corresponding nonparametric likelihood is given by

$$\prod_{t=1}^{s} F_0(u)^{\mathbb{1}\{x_t \le u\}} (1 - F_0(u))^{\mathbb{1}\{x_t > u\}}$$

where $\mathbb{1}$ is the indicator function. The maximizer of this likelihood is the empirical CDF evaluated at u: $\hat{F}_{1:s}(u) \coloneqq s^{-1} \sum_{t=1}^{s} \mathbb{1}(x_t \le u)$. Writing the log-likelihood for the entire sequence and integrating over u with a generic weight function $dw(u)$, we obtain the problem

$$\min_{t_1,\ldots,t_m} \quad -\frac{1}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \int_{-\infty}^{\infty} (t_{j+1} - t_j) \Big\{ \hat{F}_{t_j:t_{j+1}-1}(u) \log(\hat{F}_{t_j:t_{j+1}-1}(u)) + (1 - \hat{F}_{t_j:t_{j+1}-1}(u)) \log(1 - \hat{F}_{t_j:t_{j+1}-1}(u)) \Big\} \, dw(u) \ ,$$

where $\hat{F}_{t_j:t_{j+1}-1}(u) \coloneqq (t_{j+1} - t_j)^{-1} \sum_{t=t_j}^{t_{j+1}-1} \mathbb{1}(x_t \le u)$ is the empirical CDF for segment j. The dynamic programming algorithm used to solve (1.3) can be slightly modified and applied to solve this problem. Assuming the computational complexity of each integral is $O(1)$, the computational complexity remains $O(mT^2)$.

Real vectorial data vs. structured data. The normal change-in-mean model assumes that the data at each index t lies in $\mathbb{R}^d$ for some dimension d. However, often data is not vectorial but instead structured, as is the case with e.g., images and graphs. As noted in Section 1.1.3, kernels are powerful tools for dealing with structured data. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

be a kernel with corresponding canonical feature map $\phi : \mathcal{X} \to \mathcal{H}$ for some set $\mathcal{X}$ and some Hilbert space $\mathcal{H}$. The normal change-in-mean objective can be kernelized to obtain a kernel change-in-mean-element objective (Harchaoui and Cappé, 2007):

$$\min_{t_1,\dots,t_m} \quad \frac{1}{T}\sum_{j=0}^{m}\sum_{t=t_j}^{t_{j+1}-1} \|\phi(x_t) - \hat{\mu}_j\|_2^2 \,,$$

where $\hat{\mu}_j = 1/(t_{j+1}-t_j)\sum_{t=t_j}^{t_{j+1}-1}\phi(x_j)$ is the empirical mean element of segment j. This objective can be computed using the kernel trick and optimized using dynamic programming. The computational complexity is $O(mT^2)$.

Supervised vs. unsupervised. Change-point algorithms can be either supervised or unsupervised. By supervised we mean that there exist sequences in which the change points are labeled. These sequences are then used in order to learn parameters to better segment unlabeled sequences. One way to make the normal change-in-mean method a supervised method is to learn a metric. Concretely, consider a variation of the normal change-in-mean problem in which the covariance matrix Σ is the same for all observations, but is unknown and not necessarily of the form $\sigma^2 I$. Then for a fixed choice of Σ, Problem (1.3) becomes

$$\min_{t_1,\dots,t_m} \quad \frac{1}{T}\sum_{j=0}^{m}\sum_{t=t_j}^{t_{j+1}-1} (x_t - \hat{\mu}_j)^T\Sigma^{-1}(x_t - \hat{\mu}_j) \,. \tag{1.5}$$

Given change points $\mathcal{T} = \{t_1(\Sigma^{-1}), \dots, t_m(\Sigma^{-1})\}$, define the normalized segment equivalence matrix $\Pi_{\mathcal{T}} \in \mathbb{R}^{T\times T}$ by

$$[\Pi_{\mathcal{T}}]_{t,t'} = \begin{cases} \frac{1}{t_j - t_{j-1}}, & \text{if } t_{j-1}(\Sigma^{-1}) \le t, t' < t_j(\Sigma^{-1}) \text{ for some } j \\ 0, & \text{else .} \end{cases}$$

Lajugie et al. (2014) considered the setting in which n labeled sequences exist. They aimed to optimize over Σ^{-1} in order to minimize the distance between the estimated and true change

points. For a given sequence i they measured this distance in terms of the Frobenius norm difference between the estimated equivalence matrix $\hat{\Pi}^{(i)}(\Sigma^{-1})$ obtained by solving (1.5) on sequence i and the true equivalence matrix $\Pi^{(i)\star}$. The resultant problem is then

$$\min_{\Sigma^{-1}} \frac{1}{n} \sum_{i=1}^{n} \|\hat{\Pi}^{(i)}(\Sigma^{-1}) - \Pi^{(i)\star}\|_F^2 + \lambda \|\Sigma^{-1}\|_F^2 \ .$$

They optimize a convex surrogate of this objective with projected subgradient descent. Each iteration is $O(mT^2)$ and the convergence rate of the objective values is $O(1/a)$ where a is the iteration counter.

Testing vs. estimation. Finally, rather than focusing on estimation as done with the normal change-in-mean model, one can instead take a hypothesis testing approach. As previously noted, hypothesis testing and estimation are two sides of the same coin when it comes to estimating a single change point. If we do not assume that the pooled covariance matrix is the identity, then we can estimate it in the one change point case via $\hat{\Sigma}_w = \{\sum_{t=1}^{t_1-1}(x_t - \hat{\mu}_0)(x_t - \hat{\mu}_0)^T + \sum_{t=t_1}^{T}(x_t - \hat{\mu}_1)(x_t - \hat{\mu}_1)^T\}/(T-2)$. A potential change point then occurs at the index t_1 maximizing the Hotelling T^2 statistic:

$$\max_{t_1} T_{t_1}^2 \coloneqq \max_{t_1} \frac{(t_1 - 1)(T - t_1 + 1)}{T} (\hat{\mu}_0 - \hat{\mu}_1)^T \hat{\Sigma}_w^{-1} (\hat{\mu}_0 - \hat{\mu}_1) \ .$$

Srivastava and Worsley (1986) provide an improved Bonferroni bound in this setting and suggest using a binary segmentation approach to find multiple change points. If there are m change points then the computational complexity is $O(mT)$.

1.2.2 Contributions of this dissertation

The contributions of this dissertation to the change-point literature are two-fold. First, in Chapter 4 we propose a frequentist approach to learning deep feature representations for change-point estimation. Previous work in this area learned shallow features (e.g., Lajugie

et al., 2014) or learned deep features with a Bayesian approach (Nagano et al., 2019). Our approach, which is frequentist and inspired by the algorithm in Chapter 3, works with any level of supervision: the algorithm can take advantage of sequences whose change points are fully or partially-labeled. On the other hand, it can still learn feature representations in an entirely unsupervised manner. The runtime of our approach is only quadratic in the length of the sequence, as opposed to cubic, as in the method of Nagano et al. (2019).

In Chapter 5 we present an approach to change-point estimation on point clouds that is motivated by an application in oceanography. The method is based on distances between Hilbert space embeddings of empirical distributions and is an extension of the kernel change-point method of Harchaoui and Cappé (2007). We use the penalty-based approach of Lebarbier (2005) to estimate the number of change points using supplementary physical measurements.

Chapter 2

CONVOLUTIONAL KERNEL NETS AND CONVOLUTIONAL NEURAL NETS: SIMILARITIES, DIFFERENCES, AND NEW ALGORITHMS

Joint work with V. Roulet and Z. Harchaoui.[1]

Abstract. Convolutional Neural Networks, as most artificial neural networks, are commonly viewed as methods different in essence from kernel-based methods. We provide a systematic approach for transforming Convolutional Neural Networks (ConvNets) into kernel-based counterparts, Convolutional Kernel Networks (CKNs), and demonstrate that this perception can be challenged both formally and empirically. We show that, given a landmark Convolutional Neural Network, we can define a corresponding Convolutional Kernel Network, equally trainable via stochastic gradient optimization, that performs on par with its Convolutional Neural Network counterpart. We present experimental results supporting these claims on landmark ConvNet architectures, comparing each ConvNet to its CKN counterpart over several parameter settings.

2.1 Introduction

Convolutional neural networks are currently ubiquitous methods for learning feature representations from data in many computer vision and signal processing applications. The common description of a convolutional neural network consists in decomposing the architecture into layers. Each layer is a mapping performed by a parameterized function whose parameters are learned from data (Goodfellow et al., 2016). Convolutional neural networks were introduced

[1]We gratefully acknowledge support from NSF TRIPODS Award CCF-1740551, the program "Learning in Machines and Brains" of CIFAR, and faculty research awards.

and developed by Fukushima (1980); LeCun (1988, 1989) and LeCun et al. (1989, 1995, 2001), and were originally motivated by image classification problems.

A classical convolutional neural network such as one from the LeNet series (LeCun, 1988, 1989; LeCun et al., 1989, 1995, 2001) stacks two main types of layers: convolutional layers and pooling layers. These two types of layers were motivated by the Hubel-Wiesel model of human visual perception (Hubel and Wiesel, 1962). A convolutional layer decomposes into several units. Each unit is connected to local patches in the feature maps of the previous layer through a set of weights. A pooling layer computes local statistics of patches of units in individual feature maps. State-of-the-art convolutional neural networks include a variety of additions and improvements which we shall not attempt to review here (see, e.g., He et al., 2016), where our goal is to instead gain a better understanding of classical ones.

This operational description of convolutional neural networks contrasts with the mathematical description of kernel-based methods. Kernel-based methods, such as support vector machines, were at one point the most popular array of approaches for learning mappings from input examples to output labels (Schölkopf and Smola, 2002; Steinwart and Christmann, 2008). Kernels are positive-definite pairwise similarity measures that allow one to design and learn such mappings by defining them as linear functionals in a Hilbert space. Owing to the so-called reproducing property of kernels, these linear functionals can be learned from data.

This apparent antagonism between the two families of approaches is, however, misleading and somewhat unproductive. We argue and demonstrate that, in fact, *a classical convolutional neural network can be adapted or translated into a comparable convolutional kernel network*, a kernel-based architecture with an appropriate hierarchical compositional kernel. Indeed, the operational description of a ConvNet can be seen as the description of a data-dependent approximation of an appropriate kernel map. We offer a range of examples illustrating this relationship and provide empirical evidence supporting it.

The kernel viewpoint brings important insights. Despite the widespread use of convolutional neural networks, relatively little is understood about them. We would, in general, like to be able to address questions such as the following: What kinds of activation functions

should be used? How many filters should there be at each layer? Why should we use spatial pooling? Thanks to the kernel viewpoint, we can provide a statistical and computational perspective on these issues. Activation functions used in ConvNets are used to approximate a kernel, i.e., a similarity measure, between patches. The number of filters determines the quality of the approximation. Moreover, spatial pooling may be viewed as approximately taking into account the distance between patches when measuring the similarity between images.

We lay out a systematic framework to convert between a convolutional neural network (ConvNet) and a convolutional kernel network (CKN) and put it to practice with four landmark architectures on two problems. The four ConvNet architectures, LeNet-1, LeNet-5, All-CNN-C (Springenberg et al., 2015), and AlexNet (Krizhevsky et al., 2012; Krizhevsky, 2014) correspond to milestones in the development of ConvNets. We consider digit classification with LeNet-1 and LeNet-5 on MNIST (LeCun et al., 1995, 2001) and image classification with All-CNN-C and AlexNet on CIFAR-10 (Krizhevsky and Hinton, 2009) and a subset of ImageNet (Deng et al., 2009), respectively. We present an efficient algorithm to train a CKN, based on a first stage of unsupervised training and a second stage of gradient-based supervised training.

We present here a systematic experimental comparison *on an equal standing* of the two approaches on real-world data sets. By equal standing we mean that the two architectures compared are analogous from a functional viewpoint and are trained similarly from an algorithmic viewpoint. We are not aware of previous similar efforts. In contrast to previous works, the approach we adopt here is deliberately pragmatic and systematic when exploring the relationship between a ConvNet and kernel-based analogue. Indeed, we believe that a detailed mathematical description of a ConvNet, as opposed to a high-level description brushed in broad strokes, is critical to the design of a competing method from an alternative viewpoint.

The remainder of this chapter proceeds as follows. In Section 2 we highlight seminal works in this area and review previous work related to ours. Then, in Section 3, we give an

informal description of a convolutional kernel network, emphasizing the counterparts of each component of a classical convolutional neural network. This section is echoed by Appendix A.1, where we give the detailed descriptions of these components in the aforementioned classical ConvNets. These detailed descriptions are of independent interest for the history of landmark architectures in machine learning. The CKN counterpart to each ConvNet is described in Appendix A.3. In Section 4, we present general formulas of the gradient of a convolutional kernel network with respect to its parameters and a stochastic gradient algorithm to train it in a supervised manner. Finally, in Section 5, we provide experimental results demonstrating the comparable performance achieved by the proposed kernel-based counterparts and the original ConvNets we consider. The CKN code related to this chapter is publicly available in the software library *YesWeCKN* at `https://github.com/cjones6/yesweckn` .

2.2 Related Work

Three popular perspectives on the relationship between deep networks and kernels have been proposed in the literature. We describe each of these next and then proceed to discuss the broad range of related work on deep networks and kernel-based methods.

2.2.1 Kernel perspectives on deep networks

Mechanistically, a multi-layer perceptron (MLP) takes an input $x \in \mathbb{R}^{f_0}$. Given a set of weight matrices $W_1 \in \mathbb{R}^{f_1 \times f_0}, \ldots, W_L \in \mathbb{R}^{f_L \times f_{L-1}}$, a set of intercept or "bias" vectors $b_1 \in \mathbb{R}^{f_1}, \ldots, b_L \in \mathbb{R}^{f_L}$, and a nonlinear function $a : \mathbb{R} \to \mathbb{R}$, an MLP computes the output $F_{L+1}^{MLP}(x) \in \mathbb{R}^{f_L}$ for x, defined through the recursive operations

$$
\begin{aligned}
F_0^{MLP}(x) &= x \\
F_\ell^{MLP}(x) &= a(W_\ell F_{\ell-1}^{MLP}(x) + b_\ell) \ , \quad \ell = 1, \ldots, L \\
F_{L+1}^{MLP}(x) &= W_{L+1} F_L^{MLP}(x) + b_{L+1} \ ,
\end{aligned}
$$

where a is understood to be applied element-wise.

In contrast, a convolutional neural network (ConvNet) typically acts on multi-dimensional inputs. The sharing of weight vectors across spatial locations allows the learned network to achieve translation invariance. Specifically, consider an input $x \in \mathbb{R}^{f_0 \times h_0 \times w_0}$, where f_0 is the number of input channels (e.g., three for a color image with the RGB color representation) and h_0 and w_0 are the spatial dimensions. We will collapse the latter two dimensions, representing the input as $x_0 \in \mathbb{R}^{f_0 \times p_0}$. At its simplest, a convolutional network first extracts contiguous patches of total size s_1 from the input, resulting in $E_1(x_0) \in \mathbb{R}^{s_1 \times p_1}$. Then, as in an MLP, a weight matrix $W_1 \in \mathbb{R}^{f_1 \times s_1}$ (whose rows are called "filters"), bias vector $b_1 \in \mathbb{R}^{f_1}$, and nonlinearity $a : \mathbb{R} \to \mathbb{R}$ are applied. The result may then be post-processed by applying a function c_1 that acts separately on each channel (i.e., row) $1, \ldots, f_1$ in order to perform what is called "pooling". This process is repeated for some number L times, resulting in an output $F_{L+1}^{CN}(x) \in \mathbb{R}^{f_L \times p_L}$ defined recursively via

$$
\begin{aligned}
F_0^{CN}(x) &= x_0 \\
F_\ell^{CN}(x) &= c_\ell(a(W_\ell E_\ell(F_{\ell-1}^{CN}(x)) + b_\ell \mathbb{1}^T)) \;, \quad \ell = 1, \ldots, L \\
F_{L+1}^{CN}(x) &= W_{L+1} F_L(x) + b_{L+1} \mathbb{1}^T \;.
\end{aligned}
$$

Unlike an MLP, a ConvNet usually extracts multiple patches at each layer and applies the same weight matrix W_ℓ to each patch. Note that an MLP is a special case of a ConvNet in which there is only one patch at each layer, of size $f_0 h_0 w_0$ initially and f_ℓ, $\ell = 1, \ldots, L$ thereafter.

Both MLPs and ConvNets have been studied from the kernel perspective. Recall that a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for some non-empty set $\mathcal{X}$ is a positive definite kernel if there exists a Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \to \mathcal{H}$ such that for all $x, y \in \mathcal{H}$, $k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$. We now proceed to review how we can view MLPs and ConvNets from three different perspectives related to kernels: (1) random features; (2) Gaussian processes; and (3) neural tangent kernels.

Random features viewpoint. Random features, popularized by Rahimi and Recht (2007), provide a computationally efficient way to approximate kernels. From a random features viewpoint, unsupervised MLPs and ConvNets with random weights can be viewed as generating approximate feature maps of kernels based on the output from each layer. For a given layer this approximation of kernel evaluations becomes exact as the number of filters f_ℓ goes to infinity.

Concretely, first consider the case of an MLP. As noted by Daniely et al. (2016), as the number of hidden units goes to infinity, the inner product of the outputs of a single hidden layer ℓ is equal to the value of a kernel on the inputs:

Proposition 1. *Consider a measurable space Ω with probability measure μ and an activation function $a : \mathbb{R}^{f_\ell} \times \mathbb{R}^{f_\ell} \to \mathbb{R}$ such that $a(\cdot, x)$ is square integrable with respect to μ for any $x \in \mathbb{R}^{f_\ell}$. Then the pair (μ, a) defines a kernel $k : \mathbb{R}^{f_\ell} \times \mathbb{R}^{f_\ell} \to \mathbb{R}$ on inputs $x, x' \in \mathbb{R}^{f_\ell}$ as the dot product of the functions $a(\cdot, x)$ and $a(\cdot, x')$ on the measurable space Ω with probability measure μ, i.e.,*

$$k(x, x') \coloneqq \mathbb{E}_{w \sim \mu}[a(w, x)a(w, x')] \ . \tag{2.1}$$

In practice the activation function takes the form $a(w^T x)$ and we approximate the expectation with an average, obtaining an unbiased approximation of the kernel:

$$k(x, x') \approx \frac{1}{f_\ell} \sum_{c=1}^{f_\ell} a(W_{c,\cdot}^T x) a(W_{c,\cdot}^T x') \ ,$$

where the rows of W_ℓ are draws from the distribution μ. Daniely et al. (2016) proved high probability bounds on the approximation error of the inner product of the outputs of an MLP with respect to the associated compositional kernel on the inputs. Their results (which can be extended to include the case where there are biases) assume that the inputs lie on the sphere, the weights are normally distributed with a proper scaling of the variances, and the nonlinearities are nicely behaved.

In the case of convolutional networks, the analogous single-layer result applies to patches $\boxplus$ and $\boxplus'$ at each layer rather than the input features. Indeed, directly replacing x and x' by $\boxplus$ and $\boxplus'$ leads to $k(\boxplus, \boxplus') \coloneqq \mathbb{E}_{w\sim\mu}[a(w, \boxplus)a(w, \boxplus')]$. The high probability bounds of Daniely et al. (2016) apply if there is no weight sharing (i.e., the networks are locally connected rather than convolutional) and there is no pooling.

Gaussian process viewpoint. A Gaussian process is a collection of random variables, any finite subset of which has a joint Gaussian distribution (Rasmussen and Williams, 2006). Unsupervised MLPs and ConvNets can alternatively be viewed as Gaussian processes in the limit as the number of filters f_ℓ at each layer goes to infinity.

In contrast to the random features viewpoint, this viewpoint focuses on the pre-activations at each layer. That is, the terms $z_\ell \coloneqq W_\ell F_{\ell-1}(x) + b_\ell$ and $z_\ell \coloneqq W_\ell E_\ell(F_{\ell-1}(x)) + b_\ell$ in MLPs and ConvNets, respectively. Neal (1996) first observed the relationship between single-hidden-layer neural networks and Gaussian processes. Consider the case where the weights and biases are independent and normally distributed, i.e., $(W_\ell)_{ij} \sim N(0, \sigma_{W,\ell}^2/f_\ell)$ and $(b_\ell)_i \sim N(0, \sigma_{b,\ell}^2)$ for some $\sigma_{W,\ell}$ and $\sigma_{b,\ell}$ and for all $\ell = 1, \ldots, L, i = 1, \ldots, f_\ell, j = 1, \ldots, f_{\ell-1}$. The output of a single-hidden-layer MLP applied to a finite set of inputs $x^{(1)}, \ldots, x^{(n)}$ may be written as

$$\left[z_2(x^{(1)}), \ldots, z_2(x^{(n)})\right] = \left[b_2, \ldots, b_2\right] + \sum_{j=1}^{f_1} (W_2)_{\cdot,j} \left[a(z_1^{(1)})_j, \ldots, a(z_1^{(n)})_j\right] \ .$$

Observe that for a given input $x^{(i)}$, the terms $(W_2)_{\cdot,j} a(z_1^{(i)})_j$ for all j are i.i.d. conditional on the input. Assuming the activations have bounded variance, we can apply the multivariate central limit theorem to the second term. Given that the biases are normally distributed, we obtain a Gaussian process in the limit as $f_1 \to \infty$.

Since the elements of each pre-activation vector $z_2(x^{(i)})$, $i = 1, \ldots, n$ are independent conditional on the input $x^{(i)}$, we can apply similar logic at successive layers of deeper networks. Using induction, we can show that as the number of hidden units at each layer goes to infinity sequentially, we obtain a Gaussian process at each layer ℓ with mean 0 and covariance function

given by

$$[k_\ell(x^{(i)}, x^{(i')})]_{j,j'} = \begin{cases} \sigma_{b,\ell}^2 + \sigma_{W,\ell}^2 \mathbb{E}_{z_{\ell-1}\sim\mathcal{GP}(0,k_{\ell-1})}[a(z_{\ell-1}(x^{(i)}))a(z_{\ell-1}(x^{(i')}))], & j = j' \\ 0, \qquad \text{else} . \end{cases}$$

See page 4 of Lee et al. (2018). This result was strengthened by Matthews et al. (2018), who proved conditions under which this is true as the number of hidden units at each layer goes to infinity simultaneously rather than sequentially.

The story for the case of convolutional networks is similar (Garriga-Alonso et al., 2019; Novak et al., 2019). Defining the set of rows of the patches at layer ℓ corresponding to channel j by $m_{\ell,j} = \{(j-1)s_\ell/f_{\ell-1}+1, (j-1)s_\ell/f_{\ell-1}+2, \ldots, js_\ell/f_{\ell-1}\}$, the result of the convolutions at the first layer is given by

$$\left[z_1(x^{(1)}), \ldots, z_1(x^{(n)})\right] = \left[b_1\mathbb{1}^T, \ldots, b_1\mathbb{1}^T\right] + \sum_{j=1}^{f_0} (W_1)_{\cdot,m_{1,j}} \left[E_1(x^{(1)})_{m_{1,j},\cdot}, \ldots, E_1(x^{(n)})_{m_{1,j},\cdot}\right] .$$

Since the parameters $(W_1)_{ij}$ are i.i.d. Gaussian, each term in the summation is multivariate Gaussian conditional on the inputs. Therefore, since the biases are also i.i.d. Gaussian, the outputs are multivariate Gaussian as a sum of independent Gaussians. Also note that each row of the outputs $z_1(x^{(i)})$ (i.e., each feature map) is independent conditional on the input due to the independence of the weights for $i = 1, \ldots, n$.

The result of the convolutions at the second layer is then given by

$$\left[z_2(x^{(1)}), \ldots, z_2(x^{(n)})\right] = \left[b_2\mathbb{1}^T, \ldots, b_2\mathbb{1}^T\right] + \sum_{j=1}^{f_1} (W_2)_{\cdot,m_{2,j}} \left[E_2(\tilde{z}_1^{(1)})_{m_{2,j},\cdot}, \ldots, E_2(\tilde{z}_1^{(n)})_{m_{2,j},\cdot}\right] ,$$

with $\tilde{z}_1^{(i)} = c_2(a(z_1^{(i)}))$ for all $i = 1, \ldots, n$. Once again the weights and biases are i.i.d. by assumption. Moreover, since the feature maps in $z_1(x^{(i)})$ are all independent conditional on the input for $i = 1, \ldots, n$ and the pooling is applied independently to each feature map, we have that the elements in the same locations in the matrices $E_2(c_2(a(z_1^{(i)})))_{\cdot,m_{2,j}}$ and

$E_2(c_2(a(z_1^{(i)})))_{\cdot,m_{2,j'}}$ for $j \neq j'$ are independent conditional on the input. Therefore, applying the multivariate central limit theorem, we obtain a Gaussian process in the limit as $f_1 \to \infty$. Garriga-Alonso et al. (2019) show that only the diagonal elements of the covariances are required in order to compute the overall covariance at the final layer. Performing induction, we can find that as the number of filters at each layer goes to infinity sequentially, we obtain Gaussian processes at each layer with zero mean and covariances with diagonal elements

$$[k_\ell(x^{(i)}, x^{(i')})]_{j,q,j,q} = \sigma_{b,\ell}^2 + \sigma_{W,\ell}^2 \sum_{j' \in m_{\ell,j}} \mathbb{E}_{\tilde{z}_{\ell-1} \sim \mathcal{GP}(0, k_{\ell-1})}[E_2(\tilde{z}_{\ell-1}(x^{(i)}))_{j',q} E_2(\tilde{z}_{\ell-1}(x^{(i')}))_{j',q}] \ .$$

Garriga-Alonso et al. (2019) and Novak et al. (2019) delineate conditions under which this convergence holds for both the sequential and simultaneous limits.

Neural tangent kernel viewpoint. Both unsupervised and supervised MLPs and ConvNets can alternatively be viewed as generating feature maps of kernels based on the gradients of the outputs. Define $\theta = [\text{vec}(W_1), \ldots, \text{vec}(W_L), b_1^T, \ldots, b_L^T]$ and let f_θ be the dimension of θ. The neural tangent kernel is a matrix-valued kernel, defined as

$$k_{i,j}^{NTK}(x, x') = \left\langle \frac{\partial F_{L,i}(x, \theta)}{\partial \theta}, \frac{\partial F_{L,j}(x', \theta)}{\partial \theta} \right\rangle \ ,$$

where the subscripts i, j denote the ith and jth components of F_L for $i, j = 1, \ldots, f_L$, and F could be, e.g., F^{MLP} or F^{CN}. Assuming a proper scaling of the parameters, one can derive the limiting kernel in the case where the dimension of each layer goes to infinity using the Gaussian process results from above (Jacot et al., 2018; Arora et al., 2019).

Under gradient flow and/or gradient descent with a sufficiently small step size the empirical neural tangent kernel of an MLP does not change much during training (Jacot et al., 2018; Arora et al., 2019; Lee et al., 2019). For example, consider the case of the square loss and an MLP with a single output. Assume that the layer widths are equal, k^{NTK} is full rank, the inputs and outputs lie in a compact set and there are no duplicate inputs, and the

activation function is bounded, has bounded partial derivatives, and its gradient is Lipschitz. Then with a sufficiently small step size, $\sup_t \|k_t^{NTK} - k_0^{NTK}\|_F = O(n^{-1/2})$, where k_t^{NTK} denotes the value of the NTK at iteration t. In this setting it can also be shown that the evolution of the network outputs behaves like the evolution of the linearized network outputs, i.e., $\sup_t \|F_L^{MLP}(x;\theta^{(t)}) - F_L^{MLP-lin}(x;\theta^{(t)})\|_F = O(n^{-1/2})$, where $F_L^{MLP-lin}$ is the linearized network (Lee et al., 2019). In the case of convolutional networks Yang (2019) showed that the neural tangent kernel converges almost surely as the layer widths go to infinity simultaneously, as long as the nonlinearities have polynomially bounded weak derivatives.

In each of the above perspectives the results on the relationship between the MLPs/ConvNets and kernels may capture some facets of stochastic learning of deep networks but do not give a comprehensive viewpoint on the topic. The random features and Gaussian process viewpoints focus on networks with random weights. Moreover, the convergence results related to the random features viewpoint rely on the inputs lying on the sphere. In addition, the neural tangent kernel viewpoint studies the training behavior with sufficiently small step sizes. This "lazy training" regime of infinite-width networks may not reflect the reality of the training process for many networks (Chizat et al., 2019). Finally, all of the above viewpoints do not account for the behavior of input-dependent pooling (e.g., max pooling) and do not shed light on the appropriate regularization penalty to control the capacity of the networks. As noted in Novak et al. (2019), in the absence of pooling, locally connected networks converge to the same Gaussian process as convolutional networks. This suggests that the Gaussian process viewpoint may not be capturing the essence of ConvNets.

2.2.2 Contributions

This chapter develops the random features viewpoint and builds upon two interwoven threads of research related to kernels: the connections between kernel-based methods and (convolutional) neural networks and the use of compositions of kernels for the design of feature representations. The first thread dates back to Neal (1996), who showed that an

infinite-dimensional single-layer neural network is equivalent to a Gaussian process. Building on this, Williams (1996) derived what the corresponding covariance functions were for two specific activation functions. Later, Cho and Saul (2009) proposed what they termed the arc-cosine kernels, showing that they are equivalent to infinite-dimensional neural networks with specific activation functions (such as the ReLU) when the weights of the neural networks are independent and unit Gaussian. Moreover, they composed these kernels and obtained higher classification accuracy on a data set like MNIST. However, all of these works dealt with neural networks, not convolutional neural networks.

More recently, several works have proposed hybrid architectures and alternative views of convolutional neural networks. The hybrid architectures used kernel-based methods or approximations thereof as substitutes to fully-connected layers or other parts of convolutional neural networks to build hybrid ConvNet architectures (Bruna and Mallat, 2013; Huang et al., 2014; Dai et al., 2014; Yang et al., 2015; Oyallon et al., 2017, 2019). In contrast to these works, we are interested in purely kernel-based networks. Most recently, Scetbon and Harchaoui (2020) defined a kernel associated with a ConvNet based on Taylor series corresponding to the activation functions. They then examined the rate of convergence of the kernel ridge regression estimator in the associated Hilbert space to the Bayes optimal classifier.

The second thread began with Schölkopf and Smola (2002, Section 13.3.1), who proposed a kernel over image patches for image classification. The kernel took into account the inner product of the pixels at every pair of pixel locations within a patch, in addition to the distance between the pixels. This served as a precursor to later work that examined compositions of feature maps of kernels. A related idea, introduced by Bouvrie et al. (2009), entailed having a hierarchy of kernels. These kernels were defined to be normalized inner products of "neural response functions", which were derived by pooling the values of a kernel at the previous layer over a particular region of an image. In a similar vein, Bo et al. (2011) first defined a kernel over sets of patches from two images based on the sum over all pairs of patches within the sets. Within the summation is a weighted product of a kernel over the patches and a kernel over the patch locations. They then proposed using this in a hierarchical manner,

approximating the kernel at each layer by projecting onto a subspace.

Multi-layer convolutional kernels were introduced by Mairal et al. (2014b). In Mairal et al. (2014b) and Paulin et al. (2017), kernel-based methods using such kernels were shown to achieve competitive performance for particular parameter settings on image classification and image retrieval tasks, respectively. The kernels considered were however different from the kernel counterparts of the ConvNets they competed with. Recently, Shankar et al. (2020) compared the performance of an obscure family of ConvNets to their corresponding neural tangent kernels and unsupervised, non-approximated kernel networks on MNIST, variations of CIFAR-10 and CIFAR-100, and UCI datasets. Building upon the prior work by Paulin et al. (2017), Mairal (2016) proposed an end-to-end training algorithm for convolutional kernel networks. Many of these aforementioned works relied on an approximation to the kernels based on either optimization or the Nyström method (Williams and Seeger, 2000; Bo and Sminchisescu, 2009). Alternatively, kernel approximations using random Fourier features (Rahimi and Recht, 2007) or variants thereof could also approximate such kernels for a variety of activation functions (Daniely et al., 2016, 2017), although at a slower rate. Finally, Bietti and Mairal (2019) studied the invariance properties of convolutional kernel networks from a theoretical function space point of view.

Building off the work of Mairal et al. (2014b); Daniely et al. (2016, 2017); Bietti and Mairal (2019) and Paulin et al. (2017), we put to practice the transformation of a convolutional neural network into its convolutional kernel network counterpart for several landmark architectures. Transforming a convolutional neural network to its convolutional kernel network counterpart requires a careful examination of the details of the architecture, going beyond broad strokes simplifications made in previous works. When each transformation is carefully performed, the resulting convolutional kernel network can compete with the convolutional neural network. We provide all the details of our transformations in Appendix A.3. To effectively train convolutional kernel networks, we present a rigorous derivation of a general formula of the gradient of the objective with respect to the parameters. Finally, we propose a new stochastic gradient algorithm using an accurate gradient computation method to train convolutional

kernel networks in a supervised manner. As a result, we demonstrate that convolutional kernel networks perform on par with their convolutional neural network counterparts.

2.3 Convolutional Networks: Kernel and Neural Formulations

We begin this section by an informal walk through each component of a convolutional kernel network. We then proceed to describe the correspondence between each component of a CKN and the corresponding component of a ConvNet. The ConvNet and CKN architectures we use in the experiments are described in detail in Appendices A.1 and A.3.

2.3.1 Convolutional Kernel Networks

A convolutional kernel network is an architecture that stacks approximate similarity measures in order to approximate a target multi-layer similarity measure. That similarity measure is then used to perform a prediction task as usual in kernel-based methods. Convolutional kernel networks can be used on any lattice- or grid-structured data, such as sequences, signals, and images. For simplicity of exposition, we shall focus on similarity measures between images in our examples.

Similarity measure on patches

Let K measure the similarity between images F and F' as $K(F, F')$. If this similarity can be written as an inner product

$$K(F, F') = \langle \varphi(F), \varphi(F') \rangle_{\mathcal{H}_K}$$

in a space $\mathcal{H}_K$ for some mapping φ, then $\varphi(F), \varphi(F')$ can be taken as a feature representation for images F, F'. The question therefore is how to choose K for the data at hand.

An image is a set of pixel values (gray or RGB) observed on a lattice or grid. The lattice structure of images is essential. On the one hand, ignoring the lattice structure would boil down to considering the image as a set or distribution of pixel values. On the other hand,

comparing two images pixel by pixel boils down to vectorizing the two images. Practitioners have observed that the better route lies in between these two approaches and relies on the notion of a "patch" (Szeliski, 2011; Mairal et al., 2014a). A patch is a sub-image centered at a particular pixel location of an image. A patch $\boxplus^i$ of an image F consists of the pixels in the neighborhood $\mathcal{N}_i$ of the patch center $i = (x, y)$.

Let $\boxplus^i$ and $\boxplus^{i'}$, $i = 1, \ldots, m$ be patches from images F and F', respectively, where for all i, $\boxplus^i$ and $\boxplus^{i'}$ are from the same positions. Then, given a similarity measure k on the patches, we can choose the similarity measure K on the images to be given by

$$K(F, F') = \sum_{i=1}^{m} k(\boxplus^i, \boxplus^{i'}).$$

Convolutional kernel networks build such similarity measures K by using a *positive definite kernel* as the similarity measure k between patches. A positive definite kernel k implicitly maps the patches to an infinite-dimensional space (a Reproducing Kernel Hilbert Space (RKHS)) $\mathcal{H}_k$ and computes their inner product in this space, i.e., $k(\boxplus, \boxplus') = \langle \phi(\boxplus), \phi(\boxplus') \rangle_{\mathcal{H}_k}$, where ϕ is the mapping from patches to $\mathcal{H}_k$ induced by k. As long as k is a kernel, K is also a kernel and can therefore be written as $K(F, F') = \langle \varphi(F), \varphi(F') \rangle_{\mathcal{H}_K}$, where φ concatenates the information on all patches.

Network construction

If two images are similar, we would expect their patches to be similar for several patch sizes. Multi-layer CKNs incorporate this via a hierarchy of kernels. Let ϕ be the canonical feature map of a kernel k defined on patches of a given size, i.e.,

$$k(\boxplus, \boxplus') = \langle \phi(\boxplus), \phi(\boxplus') \rangle_{\mathcal{H}_k}.$$

Then $\phi(\boxplus^i)$ provides a feature representation of the ith coordinate of the image. Applying ϕ to each patch of the given size in the image, we obtain a new representation $\Phi(F)$ of an

image F (See Figure 2.1).

If two images are similar, we would expect them to also be similar when comparing their representations obtained by applying ϕ. Therefore, we may apply the same logic as before, i.e., forming patches in the new representation $\Phi(F)$ and computing their feature mapping given by another kernel. The features at each spatial location in this representation are derived from a larger portion of the original image than those in the previous representation (previous *layer*). Formally, the network consists of L layers, where at each layer ℓ, denoting by $\boxplus_\ell$ and $\boxplus'_\ell$ image patches from image representations F_ℓ and $F_\ell{}'$ output by previous layers, we apply the canonical feature map ϕ_ℓ of the kernel k_ℓ given by

$$k_\ell(\boxplus_\ell, \boxplus_{\ell'}) = \langle \phi_\ell(\boxplus_\ell), \phi_\ell(\boxplus'_\ell) \rangle_{\mathcal{H}_\ell}$$

to each patch in F_ℓ and $F_\ell{}'$.

The main feature of a CKN is to use kernels that induce non-linear feature maps as explained in Section 2.3.2. Yet, in order to increase invariance, a simple weighted cross-product kernel can be used between each layer. It computes an average of the similarity between two patches at positions i, i' in their neighborhood as

$$k_{\ell+1/2}(\boxplus_\ell^i, \boxplus_\ell^{i'}) = \sum_{j \in \mathcal{N}_i, j' \in \mathcal{N}_{i'}} \gamma_j \gamma_{j'} k_l(\boxplus_\ell^j, \boxplus_\ell^{j'}) \ ,$$

where $\mathcal{N}_i$ is a neighborhood of the coordinate i, $\gamma_j, \gamma_{j'}$ are weights assigned to the neighboring pixels, and the index $\ell + 1/2$ denotes a layer that is interleaved between layers ℓ and $\ell + 1$. The cross-product kernel corresponds to a weighted pooling of the feature maps, i.e., $k_{\ell+1/2}(\boxplus_\ell^i, \boxplus_\ell^{i'}) = \langle \phi_{\ell+1/2}(\phi_\ell(\boxplus_\ell^i)), \phi_{\ell+1/2}(\phi_\ell(\boxplus_\ell^{i'})) \rangle_{\mathcal{H}_\ell}$ where

$$\phi_{\ell+1/2}(\phi_\ell(\boxplus_\ell^i)) = \sum_{j \in \mathcal{N}_i} \gamma_j \phi_\ell(\boxplus_\ell^j) \ .$$

For example, for average pooling, $\gamma_j = 1/|\mathcal{N}_i|$.

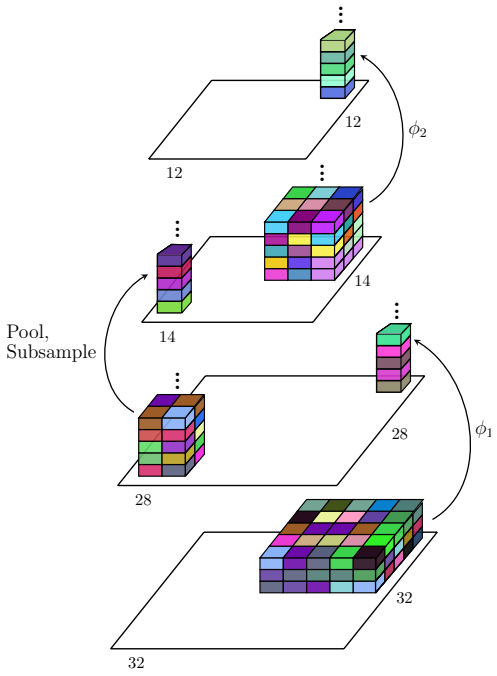


Figure 2.1: Example CKN architecture. The block sizes are the sizes of the filters at each layer. The height of the block at the input layer is three to represent the input having three channels. At every subsequent layer the feature representation is infinite-dimensional. The arrows indicate how one block gets transformed into the block at the next layer. The numbers on the sides of the parallelograms indicate the spatial dimensions of the feature representations at each layer. The colors represent the different values of the elements within the feature representations.

After pooling we often subsample the locations in the image for computational purposes. Subsampling by an integer factor of k entails retaining every kth feature representation in each row and then every kth feature representation in each column. By subsampling after pooling we aim to remove redundant features. Building layers in the above manner by applying feature maps ϕ_ℓ, pooling, and subsampling, we obtain a convolutional kernel network.

Example 1. *Figure 2.1 depicts an example CKN. In the figure an initial RGB image of size* 32×32 *(represented by the bottom rhombus in the figure) gets transformed by applying feature map* ϕ_1 *to patches of size* 5×5*. As* ϕ_1 *is applied with stride* 1×1 *(i.e., it is*

applied to every possible contiguous 5×5 *patch in the image), this results in a new feature representation (second rhombus) with spatial dimensions* 28×28*. Atop each spatial location sits an infinite-dimensional vector.*

At the second layer 2×2 *pooling is applied to the infinite-dimensional vectors, followed by subsampling by a factor of 2. The pooling is performed on all contiguous* 2×2 *patches, which initially decreases the spatial dimensions to* 27×27*. Subsampling by a factor of two entails removing the features on top of every other spatial location, yielding an output with spatial dimensions* 14×14*. The output of this layer results in the next representation (third rhombus).*

Finally, the figure depicts the application of another feature map ϕ_2 *to patches of size* 3×3 *to obtain the feature representation at the final layer. As the stride is* 1×1*, this results in an output with spatial dimensions* 12×12*.*

Given such a network, we may then compute the similarity of two images by concatenating each image's feature representation at the final layer and then applying a linear kernel. While there are only two feature maps ϕ_ℓ*,* $\ell = 1, 2$*, depicted in this figure, the process could continue for many more layers.*

Extracting feature maps

While computing the overall kernel exactly is theoretically possible (assuming that the kernels at each layer only depend on inner products of features at previous layers), it is computationally unwieldy for even moderately-sized networks. To overcome the computational difficulties, we approximate the kernel k_ℓ at each layer ℓ by finding a finite-dimensional map ψ_ℓ such that $k_\ell(\boxplus_\ell, \boxplus'_\ell) \approx \langle \psi_\ell(\boxplus_\ell), \psi_\ell(\boxplus'_\ell) \rangle_{\mathbb{R}^{f_\ell}}$ for some positive integer f_ℓ. The ψ_ℓ's then replace the ϕ_ℓ's at each layer, thereby providing feature representations of size f_ℓ of the patches at each layer ℓ. There are many ways to choose ψ_ℓ, including directly optimizing an approximation to the kernel, using random features, and projecting onto a subspace.

We consider here the approximation resulting from the projection onto a subspace spanned by "filters", usually referred to as the Nyström method (Williams and Seeger, 2000; Bo and

Sminchisescu, 2009; Mairal, 2016). These filters may be initialized at random by sampling patches from the images. We shall show in Sections 2.4.1-2.4.2 how to differentiate through this approximation and learn the filters from data in a supervised manner.

Consider a dot product kernel k with corresponding RKHS $\mathcal{H}$ and canonical feature map ϕ. Furthermore, let $w_1, \ldots, w_f \in \mathbb{R}^s$ be a set of *filters*. These filters may be selected in a number of ways, including randomly sampling from a set of patches, running k-means on a set of patches and using the resultant centroids, learning them to best approximate the Gram matrix on the inputs, and learning them using an end-to-end approach (see Section 2.4). Given a patch $\boxplus \in \mathbb{R}^s$ of size s, the Nyström approximation projects $\phi(\boxplus)$ onto the subspace spanned by $\phi(w_1), \ldots, \phi(w_f)$ in $\mathcal{H}_k$ by solving the kernel least squares problem

$$\alpha^\star \in \operatorname*{arg\,min}_{\alpha \in \mathbb{R}^f} \left\| \phi(\boxplus) - \sum_{i=1}^{f} \alpha_i \phi(w_i) \right\|_{\mathcal{H}}^2 .$$

Defining $W = [w_1, \ldots, w_f]^T \in \mathbb{R}^{f \times s}$ and assuming that k is a dot product kernel, this results in the coefficients $\alpha^\star = k(WW^T)^{-1} k(W\boxplus)$, where the kernel k is understood to be applied element-wise.[2] Therefore, for two patches $\boxplus$ and $\boxplus'$ with corresponding optimal coefficients α and α', we have

$$\begin{aligned}
\langle \phi(\boxplus), \phi(\boxplus') \rangle_{\mathcal{H}} &\approx \left\langle \sum_{i=1}^{f} \alpha_i^\star \phi(w_i), \sum_{i=1}^{f} \alpha_i^{\star\prime} \phi(w_i) \right\rangle_{\mathcal{H}} \\
&= k(W\boxplus)^T k(WW^T)^{-1} k(W\boxplus') \\
&= \langle k(WW^T)^{-1/2} k(W\boxplus), k(WW^T)^{-1/2} k(W\boxplus') \rangle_{\mathbb{R}^f} .
\end{aligned}$$

Hence, a finite-dimensional approximate feature representation of $\boxplus$ is given by

$$\psi(\boxplus) = (k(WW^T))^{-1/2} k(W\boxplus) \in \mathbb{R}^f .$$

---

[2] A dot product kernel is a kernel of the form $k(x, y) = f(\langle x, y \rangle)$ for a function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. For notational convenience for a dot product kernel k we will write $k(t)$ rather than $k(x, y)$ where $t = \langle x, y \rangle$. For a matrix A the element-wise application of k to $A \in \mathbb{R}^{m \times n}$ results in $k(A) \coloneqq [k(A_{i,j})]_{i,j=1}^{m,n}$.

We will add a regularization term involving a small value $\epsilon > 0$, as $k(WW^T)$ may be poorly conditioned. Since optimizing over W leads to a non-convex problem, prior work has studied the error of this approximation when setting W using, e.g., k-means++ (Oglic and Gärtner, 2017). See also Rudi and Rosasco (2017) for a theoretical comparison of the performance of random features and the Nyström method for ridge regression.

Denote the input features to layer ℓ by $F_{\ell-1}$, where the rows index the features and the columns index the spatial locations (which are flattened into one dimension). Let E_ℓ be a function that extracts patches from $F_{\ell-1}$. We then write the features output by the Nyström method as

$$F_\ell = \Psi_\ell(F_{\ell-1}, W_\ell) = (k(WW^T) + \epsilon I)^{-1/2} k(W E_\ell(F_{\ell-1})).$$

Here Ψ_ℓ denotes the function that applies the approximate feature map ψ_ℓ as derived above to the features at each spatial location. It is important to note that the number of filters f_ℓ controls the quality of the approximation at layer ℓ. Moreover, such a procedure results in the term $W_\ell E_\ell(F_{\ell-1})$, in which the filters are *convolved* with the images. This convolution is followed by a *non-linearity* computed using the kernel k, resulting in the application of Ψ_ℓ.

Overall formulation

The core hyperparameter in CKNs is the choice of kernel. For simplicity of the exposition we assume that the same kernel is used at each layer. Traditionally, CKNs use normalized kernels of the form

$$k(\boxplus, \boxplus') = \| \boxplus \|_2 \| \boxplus' \|_2 \tilde{k}(\boxplus^\top \boxplus' / \| \boxplus \|_2 \| \boxplus' \|_2)$$

where $\tilde{k}$ is a dot-product kernel on the sphere. Examples of such kernels $\tilde{k}$ include the arc-cosine kernel of order 0 and the RBF kernel on the sphere. Here we allow for this formulation. Using dot-product kernels on the sphere allows us to restrict the filters to lying on the sphere.

Doing so adds a projection step in the optimization.

Let F_0 be an input image. Denote by W_ℓ the filters at layer ℓ, E_ℓ the function that extracts patches from F_ℓ at layer ℓ, and N_ℓ the function normalizing the patches of F_ℓ at layer ℓ. Furthermore, let P_ℓ be the pooling and subsampling operator, represented by a matrix. (See Appendix A.4 for precise definitions.) Then the representation at the next layer given by extracting patches, normalizing them, projecting onto a subspace, re-multiplying by the norms of the patches, pooling, and subsampling is given by

$$F_\ell = \Psi_\ell(F_{\ell-1}, W_\ell) \coloneqq (k(W_\ell W_\ell^T) + \epsilon \, \mathrm{I})^{-1/2} k(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_\ell)^{-1}) N_\ell(F_{\ell-1}) P_\ell \; .$$

After L such compositions we obtain a final representation of the image that can be used for a classification task.

Precisely, given a set of images $F^{(1)}, \ldots, F^{(n)}$ with corresponding labels $y^{(1)}, \ldots, y^{(n)}$, we consider a linear classifier parametrized by W_{L+1} and a loss $\mathcal{L}$, leading to the non-convex optimization problem

$$\begin{aligned} \min_{W_1, \ldots, W_{L+1}} \quad & \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(y^{(i)}, \left\langle W_{L+1}, F_L^{(i)} \right\rangle\right) + \lambda \|W_{L+1}\|_F^2 \\ \text{subject to} \quad & W_\ell \in S^{d_\ell} \quad \text{for } \ell = 1, \ldots, L \; . \end{aligned} \tag{2.2}$$

Here $\lambda \geq 0$ is a regularization parameter for the classifier and S^{d_ℓ} is the product of Euclidean spheres in which the filters W_ℓ at the layer ℓ lie; see Appendix A.4.

2.3.2 Connections to ConvNets

A CKN may be viewed as an infinite-dimensional analogue of a ConvNet. Table 2.1 lists a set of transformations between ConvNets and CKNs. These are discussed below in more detail. For the remainder of this section we let $G \in \mathbb{R}^{f \times s_1 \times s_2}$ denote the feature representation of an image in a ConvNet. For clarity of exposition we represent it as a 3D tensor rather than a 2D matrix as for CKNs above. Here the first dimension indexes the features while the second

ConvNet component	CKN component
Convolutional layer	Projection onto the same subspace for all patch locations
Partially-connected layer	Projection onto a different subspace for each region
Fully-connected layer	Projection onto a subspace for the entire image representation
Convolution + no nonlinearity	Applying feature map of linear kernel
Convolution + tanh nonlinearity	Applying feature map of arc-cosine kernel of order 0*
Convolution + ReLU nonlinearity	Applying feature map of arc-cosine kernel of order 1
Average pooling	Applying feature map of a cross-product kernel
Max pooling	Applying feature maps of max kernels
Local response normalization	Dividing patches by their norm*

* denotes an inexact correspondence.

Table 2.1: Correspondences between ConvNets and CKNs

and third dimensions index the spatial location. We denote the element of G in feature map z at spatial location (x, y) by $(G)_{z,x,y}$.

Convolution and activation function

The main component of ConvNets is the convolution of patches with filters, followed by a pointwise nonlinearity. More precisely, denote the filters by $W \in \mathbb{R}^{f \times s}$, a patch from G by $\boxplus \in \mathbb{R}^s$, and a nonlinearity by $a : \mathbb{R}^s \times \mathbb{R}^s \to \mathbb{R}$. A ConvNet computes $a(W\boxplus)$ for every patch $\boxplus$ in G where a is understood to be applied element-wise. This can be seen as an approximation of a kernel, as stated in Proposition 1. Hence, the convolution and pointwise nonlinearity in ConvNets with random weights approximate a kernel on patches. This approximation converges to the true value of the kernel as the number of filters f goes to infinity. The downside to using such a random feature approximation is that it produces less concise approximations of kernels than, e.g., the Nyström method. In order to assess whether trained CKNs perform similarly regardless of the approximation, we approximate CKNs using the Nyström method.

Several results have been proven relating specific activation functions to their corresponding kernels. For example, the ReLU corresponds to the arc-cosine kernel of order 1 (Cho and Saul, 2009) and the identity map corresponds to the linear kernel. The tanh nonlinearity

may be approximated by a step function, and a step function corresponds to the arc-cosine kernel of order 0 (Cho and Saul, 2009).

Layer type

A ConvNet may have several types of layers, including convolutional, partially-connected, and fully connected layers. Each layer is parameterized by filters. Convolutional layers define patches and apply the same set of filters to each patch. On the other hand, partially-connected layers in ConvNets define patches and apply filters that differ across image regions to the patches. Finally, fully connected layers in ConvNets are equivalent to convolutional layers where the size of the patch is the size of the image.

Like a ConvNet, a CKN may have convolutional, partially-connected, and fully connected layers. Recall from Section 2.3.1 that a CKN projects onto a subspace at each layer and that the subspace is defined by a set of filters. At convolutional layers in a CKN, the projection is performed onto the same subspace for every patch location. On the other hand, for partially connected layers for a CKN, the projection is performed onto a different subspace for each image region. Finally, for fully connected layers a CKN projects onto a subspace defined by filters that are the size of the feature representation of an entire image.

Pooling

The pooling in a ConvNet can take many forms, including average pooling and max pooling. In each case, one defines spatial neighborhoods within the dimensions of the current feature representation (e.g., all 2×2 blocks). Within each neighborhood a local statistic is computed from the points within each feature map.[3] Concretely, for a spatial neighborhood $\mathcal{N}$ centered

[3]In the ConvNet literature, in contrast to the kernel literature, a feature map is defined as a slice of the feature representation along the depth dimension. That is, for a given z, $(G)_{z,\cdot,\cdot}$ is a feature map.

at the point (x, y), average pooling computes

$$(\tilde{G})_{z,x,y} = \frac{1}{|\mathcal{N}|} \sum_{(x',y') \in \mathcal{N}} (G)_{z,x',y'}$$

for all $z = 1, \ldots, f$.

As shown above, average pooling is the feature map associated with a cross-product kernel. Note that Mairal et al. (2014b); Mairal (2016) and Paulin et al. (2017) considered Gaussian pooling in the CKNs. In this formulation, the weight of a feature map at location x' when averaging about a feature map at location x is given by $\exp(-\|x' - x\|_2^2/(2\sigma^2))$, where σ is a hyperparameter.

Max pooling in a ConvNet does not take an average but the maximum of the local statistic. For a spatial neighborhood $\mathcal{N}$ centered at the point (x, y), max pooling computes

$$(\tilde{G})_{z,x,y} = \max_{(x',y') \in \mathcal{N}} |G|_{z,x',y'}$$

for all $z = 1, \ldots, f$. This defines a feature map and therefore a kernel on the patches. Formally, vectorizing the neighborhood of the feature representation at coordinates i, i' as $g, g' \in \mathbb{R}^{s^2 \times f}$ where s^2 is the size of the neighborhood, max pooling corresponds to the feature map $\psi(g) = (\max_{i \in \{1,\ldots,s^2\}} |g_{i,z}|)_{z=1,\ldots,f}$. The resulting kernel then writes as

$$k(g, g') = \langle \psi(g), \psi(g') \rangle_{\mathbb{R}^f} \ .$$

Normalization

There are a wide range of normalizations that have been proposed in the ConvNet literature. Normalizations of ConvNets modify the representation at each location (z, x, y) of G by taking into account values in a neighborhood $\mathcal{N}$ of (z, x, y). One such normalization is local

response normalization. Local response normalization computes

$$(\tilde{G})_{z,x,y} = \frac{(G)_{z,x,y}}{\left(\alpha + \beta \sum_{(z',x',y') \in \mathcal{N}} (G)^2_{z',x',y'}\right)^{\gamma}}$$

for all z, x, y where α, β, and γ are parameters that can be learned. In local response normalization the neighborhood $\mathcal{N}$ is typically defined to be at a given spatial location (x, y) across some or all of the feature maps. However, the spatial scale of the neighborhood could be expanded to be defined across multiple locations within feature maps.

In CKNs there is not a meaningful counterpart to defining a neighborhood across only a subset of the feature maps. Therefore, we present only a counterpart to using all feature maps at once. Consider local response normalization in ConvNets when taking the neighborhood to be the locations across all feature maps within a given spatial area. This roughly corresponds to dividing by a power of the norm of a patch in CKNs when $\alpha = 0$ and $\beta = 1$.

2.4 Supervised Training of Convolutional Kernel Networks

Like any functional mapping defined as a composition of modules differentiable with respect to their parameters, a CKN can be trained using a gradient-based optimization algorithm. An end-to-end learning approach to training a CKN in a supervised manner was first considered by Mairal (2016). We give here the full derivation of general formulas of the gradient of the training objective with respect to the CKN parameters, as well as an efficient numerical procedure to compute the gradient. We then present a stochastic gradient training algorithm to train a CKN in a supervised manner such that its performance is comparable to its ConvNet counterpart.

2.4.1 Gradient of the training objective with respect to the weights

When a CKN uses a differentiable dot product kernel k, each layer of the CKN is differentiable with respect to its weights and inputs. Therefore, the entire CKN is differentiable. This provides a benefit over commonly-used ConvNets that use non-differentiable activation

functions such as the ReLU, which must be trained using subgradient methods rather than gradient methods. Note that, while widely used, only weak convergence guarantees are known for stochastic subgradient methods. Moreover, they require sophisticated topological non-smooth analysis notions (Davis et al., 2019). As we shall show here, a CKN with a kernel corresponding to a smooth nonlinearity performs comparably to a ConvNet with non-smooth nonlinearities. We would like to underscore that, contrary to popular beliefs perhaps fueled by some choices made by Krizhevsky et al. (2012), many kinds of activations functions other than ReLU activation functions have been used with great success by practitioners; see Eger et al. (2018) for a recent account.

The derivatives of the loss function $\mathcal{L}$ from Section 2.3.1 with respect to the filters at each layer and the inputs at each layer can be derived using the chain rule. First recall the output of a single convolutional layer presented in Section 2.3.1:

$$F_\ell = \Psi_\ell(F_{\ell-1}, W_\ell) \coloneqq (k(W_\ell W_\ell^T) + \epsilon I)^{-1/2} k(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1}) N_\ell(F_{\ell-1}) P_\ell \,. \tag{2.3}$$

We then have the following proposition, which is detailed in Appendix A.4.

Proposition 2. *Let $\mathcal{L}(y, \langle W_{L+1}, F_L\rangle)$ be the loss incurred by an image-label sample (F_0, y), where F_L is the output of Lth layer of the network described by (2.3) and W_{L+1} parameterizes the linear classifier. Then the Jacobian of the loss with respect to the inner weights W_ℓ, $1 \le \ell \le L$ is given by*

$$\nabla_{\mathrm{vec}(W_\ell)} \mathcal{L}\left(y, \langle W_{L+1}, F_L\rangle\right) = \mathcal{L}' \,\mathrm{vec}(W_{L+1})^T \left[\prod_{\ell'=\ell+1}^{L} \nabla_{\mathrm{vec}\left(F_{\ell'-1}\right)} \mathrm{vec}\left(\Psi_{\ell'}\right)\right] \nabla_{\mathrm{vec}(W_\ell)} \mathrm{vec}\left(\Psi_\ell\right) \,,$$

where $\mathcal{L}' = \frac{\partial \mathcal{L}(\bar{y},\hat{y})}{\partial \hat{y}}|_{(\bar{y},\hat{y})=(y,\langle W_{L+1}, F_L\rangle)}$, and $\nabla_{\mathrm{vec}(F_{\ell'-1})} \mathrm{vec}(\Psi_{\ell'})$ and $\nabla_{\mathrm{vec}(W_\ell)} \mathrm{vec}(\Psi_\ell)$ are detailed in Propositions 33 and 27, respectively.

Computing the derivatives of the output of a convolutional layer involves several linear algebra manipulations. The critical component lies in differentiating through the matrix inverse square root. For this we make use of the following lemma.

Lemma 3. *Define the matrix square root function* $g : S_{++}^n \to \mathbb{R}^{n\times n}$ *by* $g(A) = A^{1/2}$. *Then for a positive definite matrix* $A \in S_{++}^n$ *and a matrix* $H \in \mathbb{R}^{n\times n}$ *such that* $A + H \in S_{++}^n$ *we have*

$$\operatorname{vec}(g(A+H)) = \operatorname{vec}(g(A)) + \left(\mathrm{I}_n \otimes A^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \operatorname{vec}(H) + o(\|H\|_F) \ .$$

Hence, computing the gradient of the CKN in this manner consists of solving a continuous Lyapunov equation (Khalil, 1992, Chapter 3.3). The remainder of the gradient computations involve Kronecker products and matrix multiplications; see Appendix A.4.

An upper bound β_L on the Lipschitz constant of the gradient of a CKN is defined through the recursion

$$\begin{aligned} \beta_0 &= 0 \\ \beta_\ell &= \alpha_{\ell-1}\beta_\ell^F + \alpha_\ell^F \beta_{\ell-1} + \mathcal{E}_\ell \beta_\ell^W \ , \qquad \ell = 1, \ldots, L \ , \end{aligned}$$

where α_ℓ is defined in Proposition 39 in Appendix A.5.1, α_ℓ^F, β_ℓ^F, and β_ℓ^W are defined in Appendix A.5.3, $W = (\operatorname{vec}(W_1), \ldots, \operatorname{vec}(W_L))$, and $\mathcal{E}_\ell \coloneqq \nabla_W \operatorname{vec}(W_\ell)$ is a matrix with ones in the entries for which the row and column correspond to the same element of W_ℓ and zeros elsewhere. The details of the quantities involved, along with the proof, may be found in Appendix A.5.

This bound informs the architecture design as well as the scaling of step sizes involved in gradient-based optimization. The upper bound scales roughly at least exponentially in the number of layers L. A driving component of the growth is the scaling $M_\ell \propto \|P_\ell\|_F M_k M_{\ell-1}$, if the global bound is unwrapped into its elementary components, as described in Appendix A.5. Here M_ℓ is an upper bound on the norm of the features from a CKN at layer ℓ and M_k is an upper bound on the output of the kernel k. For an architecture with a cuboid shape, that is, an architecture with layers having all the same properties, the upper bound grows exponentially as the network depth grows, driven by both the patch related terms $\|P_\ell\|_F$

Algorithm 1 Intertwined Newton Method for Matrix Inverse Square Root

Input: Positive definite matrix $M \in \mathbb{R}^{d\times d}$
Number of iterations $t_{\max}$
Initialize: $S_0 = \|M\|_F^{-1} M$, $T_0 = \mathrm{I}_d$
for $t = 1, \ldots, t_{\max}$ **do**
  $S_{t+1} \leftarrow \frac{1}{2} S_t \left(3\,\mathrm{I}_d - T_t S_t\right)$
  $T_{t+1} \leftarrow \frac{1}{2} \left(3\,\mathrm{I}_d - T_t S_t\right) T_t$
end for
$T \leftarrow \|M\|_F^{-1/2} T_{t_{\max}}$
Output: T (the approximate matrix inverse square root of M)

and the general terms $\sqrt{p_\ell s_\ell} f_\ell M_k$. This sheds light on architecture design. Indeed, for an architecture with a pyramidal shape, the growth of the upper bound as the network depth grows can be mitigated by taking smaller and smaller patch sizes and numbers of filters as the layers are overlaid.

2.4.2 Differentiating through the matrix inverse square root

The straightforward approach to computing the derivative of the matrix inverse square root involved in Proposition 2 is to call a solver for continuous-time Lyapunov equations. However, this route becomes an impediment for large-scale problems requiring fast matrix-vector computations on GPUs. An alternative is to leave it to current automatic differentiation software, which computes it through a singular value decomposition. This route does not leverage the structure and leads to worse estimates of gradients (See Section 2.5.2).

Here we propose a simple and effective approach based on two intertwined Newton methods. Consider the matrix $M = k\left(WW^T\right) + \epsilon\,\mathrm{I}_f \in S_{++}^f$. We aim to compute $M^{-1/2}$ by an iterative method. Denote by $\lambda_1, \ldots, \lambda_f$ the eigenvalues of M and let

$$H = \begin{pmatrix} 0 & M \\ \mathrm{I}_d & 0 \end{pmatrix}.$$

As the eigenvalues of H are $(\pm\sqrt{\lambda_i})_{i=1,\ldots,f}$, H can be diagonalized as $H = U\Lambda U^{-1}$ with Λ the

diagonal matrix of its eigenvalues. Let $\text{sign}(H) = U\,\text{sign}(\Lambda)U^{-1}$, where $\text{sign}(\Lambda)$ is a diagonal matrix whose diagonal is the sign of the eigenvalues in Λ. Then $\text{sign}(H)$ satisfies (Higham, 2008, Theorem 5.2)

$$\text{sign}(H) = \begin{pmatrix} 0 & M^{1/2} \\ M^{-1/2} & 0 \end{pmatrix}.$$

The sign matrix of H is a square root of the identity, i.e., $\text{sign}(H)\,\text{sign}(H) = \mathrm{I}_f$. It can then be computed by a Newton's method starting from $X_0 = H$ followed by $X_{t+1} = (X_t + X_t^{-1})/2$. Provided that $\|H\|_2 \leq 1$, it converges quadratically to $\text{sign}(H)$ (Higham, 2008, Theorem 5.6). Decomposing the iterates of this Newton's method on the blocks defined in H gives Denman and Beavers' algorithm (Denman and Beavers Jr., 1976). This algorithm begins with $S_0 = M$ and $T_0 = \mathrm{I}_f$ and proceeds with the iterations $S_{t+1} = (S_t + T_t^{-1})/2$ and $T_{t+1} = (T_t + S_t^{-1})/2$. The sequence T_t then converges to $M^{-1/2}$.

Each iteration, however, involves the inverses of the iterates T_t and S_t, which are expensive to compute when using a large number of filters. We propose applying the Newton method one more time, yet now to compute T_t^{-1} and S_t^{-1} (sometimes called the Newton-Schulz method), starting respectively from S_t and T_t as initial guesses (Higham, 1997). An experimental evaluation of this strategy when we run, say, 20 iterations of the outer Newton method (to compute the inverse matrix square root) yet only 1 iteration of the inner Newton method (to compute the inverse matrices) demonstrates that it is remarkably effective in practice (See Figure 2.2 in Section 2.5.2). We present the pseudocode in Algorithm 1 for the case where one iteration of the inner Newton's method is used. Note that we first scale the matrix M by its Frobenius norm to ensure convergence.

By differentiating through these iterations we can obtain the derivatives of $M^{-1/2}$ with respect to the entries of M. Comparing the accuracy of the gradient obtained using this algorithm to the one returned using automatic differentiation in PyTorch, we find that our approach is twice as accurate. Furthermore, the algorithm only involves matrix multiplications, which is critical to scale to large problems with GPU computing. Hence, this provides a superior means of computing the gradient.

Algorithm 2 Supervised Training of CKNs

Input: Uninitialized CKN with L layers
  Inputs $F_0^{(i)} \in \mathbb{R}^{f_0 \times p_0'}$ and labels $y^{(i)}$, $i = 1, \ldots, n$
  Number of iterations of alternating minimization to perform, T
Initialize: Do unsupervised training of the filters $W_1, \ldots, W_L$ with spherical k-means.
    Compute the features $F_L^{(i)}$ for all inputs i.
    Train the weights W_{L+1} of the classifier using a quasi-Newton method.
Supervised training:
for $j = 1, \ldots, T$ **do**
  Compute the features $F_L^{(i)}$ for a mini-batch of inputs.
  Perform one step of the Ultimate Layer Reversal method (see Algorithm 3).
end for
Compute the features $F_L^{(i)}$ for all inputs i.
Train the weights W_{L+1} of the classifier using a quasi-Newton method.
Output: $W_1, \ldots, W_L, W_{L+1}$ (the optimal filters and classifier weights)

2.4.3 Training procedures

The training of CKNs consists of two main stages: unsupervised and supervised learning. The unsupervised learning entails first initializing the filters with an unsupervised method and then fixing the filters and optimizing the ultimate layer using the full data set. The supervised learning entails training the whole initialized architecture using stochastic estimates of the objective. Here we detail the second stage, for which we propose a new approach. Algorithm 2 outlines the overall CKN training with this new method.

Stochastic gradient optimization on manifolds

A major difference between CKNs and ConvNets is the spherical constraints imposed on the inner layers. On the implementation side, this simply requires an additional projection during the gradient steps for those layers. On the theoretical side it amounts to a stochastic gradient step on a manifold whose convergence to a stationary point is still ensured, provided that the classifier is not regularized but constrained. Specifically, given image-label pairs

$(F_0^{(i)}, y^{(i)})_{i=1}^n$, we consider the constrained empirical risk minimization problem

$$\begin{aligned} \min_{W_1,\dots,W_{L+1}} \quad & \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(y^{(i)}, \left\langle W_{L+1}, F_L^{(i)} \right\rangle\right) \\ \text{subject to} \quad & W_\ell \in S^{d_\ell} \qquad\qquad \text{for } \ell = 1,\dots,L \\ & \|W_{L+1}\|_F \le \lambda \,, \end{aligned} \tag{2.4}$$

where S^{d_ℓ} is the product of Euclidean unit spheres at the ℓth layer and $F_L^{(i)}$ is the output of L layers of the network described by (2.3). Projected stochastic gradient descent draws a mini-batch B_t of samples at iteration t, forming an estimate f_{B_t} of the objective, and performs the following update:

$$\begin{aligned} W_\ell^{(t+1)} &= \mathrm{Proj}_{S^{d_\ell}}\left(W_\ell^{(t)} - \gamma_t \nabla_{W_\ell} f_{B_t}(W^{(t)})\right) \qquad \text{for } \ell = 1,\dots,L \\ W_{L+1}^{(t+1)} &= \mathrm{Proj}_{\mathcal{B}_{2,\lambda}}\left(W_{L+1}^{(t)} - \gamma_t \nabla_{W_{L+1}} f_{B_t}(W^{(t)})\right), \end{aligned} \tag{SGO}$$

where $\mathcal{B}_{2,\lambda}$ is the Euclidean ball centered at the origin of radius λ and γ_t is a step size. Its convergence is stated in the following proposition and is detailed in Appendix A.6.

Proposition 4. *Assume the loss in the constrained training problem* (2.4) *and the kernel defining the network* (2.3) *are continuously differentiable. Projected stochastic gradient descent with step size* $\gamma_t = c/\sqrt{T}$*, where* $c > 0$ *and* T *is the maximum number of iterations, finds an* $O(1/\sqrt{T})$*-stationary point.*

In practice we use the penalized formulation to compare with classical optimization schemes for ConvNets.

Ultimate layer reversal

The network architectures present a discrepancy between the inner layers and the ultimate layer: the former computes a feature representation, while the latter is a simple classifier that could be optimized easily once the inner layers are fixed. This motivates us to back-propagate

the gradient in the inner layers *through the classification performed in the ultimate layer*. Formally, consider the regularized empirical risk minimization problem (2.2),

$$\min_{W\in\mathcal{C},V} \quad f(W,V) \coloneqq \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(y^{(i)}, \left\langle V, F_L^{(i)}(W)\right\rangle\right) + \lambda\|V\|_F^2$$

where $W = (W_1, \dots, W_L)$ denotes the parameters of the inner layers constrained on spheres in the set $\mathcal{C}$, $V = W_{L+1}$ parameterizes the last layer and $F_L^{(i)}(W)$, $i = 1, \dots, n$ are the feature representations of the images output by the network. The problem can be simplified as

$$\min_{W\in\mathcal{C},V} f(W,V) = \min_{W\in\mathcal{C}} \hat{f}(W) \qquad \text{where} \qquad \hat{f}(W) \coloneqq \min_{V} f(W,V)\ .$$

Strong convexity of the classification problem ensures that the simplified problem is differentiable and its stationary points are stationary points of the original objective. This is recalled in the following proposition, which is detailed in Appendix A.7.

Proposition 5. *Assume that $f(W,V)$ is twice differentiable and that for any W, the partial functions $V \to f(W,V)$ are strongly convex. Then the simplified objective $\hat{f}(W) = \min_V f(W,V)$ is differentiable and satisfies*

$$\|\nabla \hat{f}(W)\|_2 = \|\nabla f(W, V^*)\|_2\ ,$$

where $V^* = \arg\min_V f(W,V)$.

Therefore, if a given W^* is ϵ-near stationary for the simplified objective $\hat{f}$, then the pair $(W^*, V^*(W))$, where $V^*(W) = \arg\min_V f(W,V)$, is ϵ-near stationary for the original objective f.

Square loss. In the case of the square loss, the computations can be performed analytically, as shown in Appendix A.7. However, the objective cannot be simplified on the whole data set, since it would lose its decomposability in the samples. Instead, we apply this strategy on mini-batches. I.e., at iteration t, denoting f_{B_t} the objective formed by a mini-batch B_t of the

samples, the algorithm updates the inner layers via, for $\ell = 1, \dots L$,

$$W_\ell^{(t+1)} = \mathrm{Proj}_{S^{d_\ell}} \left(W_\ell^{(t)} - \gamma_t \, \mathrm{Proj}_{S^{d_\ell}} \left(\nabla_{W_\ell} \hat{f}_{B_t}(W^{(t)}) \right) \right)$$

where $\hat{f}_{B_t}(W^{(t)}) = \min_V f_{B_t}(W^{(t)}, V)$ and we normalize the gradients by projecting them on the spheres to use a single scaling for all layers.

Other losses. For other losses such as the multinomial loss, no analytic form exists for the minimization. At each iteration t we therefore approximate the partial objective $g_{B_t}(\cdot; W) : V \to f_{B_t}(W, V)$ on the mini-batch B_t by a regularized quadratic approximation and perform the step above on the inner layers. The ultimate layer reversal step at iteration t is detailed in Algorithm 3. The quadratic approximation $q_{g_{B_t}}^{(t)}(V; W_{1:L})$ in Step 7 depends on the current point $V^{(t)}$ and can be formed using the full Hessian or a diagonal approximation of the Hessian. The gradient in Step 9 is computed by back-propagating through the operations.

2.5 Experiments

In the experiments we seek to address the following two questions:

1. How well do the proposed training methods perform for CKNs?

2. Can a supervised CKN attain the same performance as its ConvNet counterpart?

Previous works reported that specially-designed CKNs can achieve comparable performance to ConvNets in general on MNIST and CIFAR-10 (Mairal et al., 2014b; Mairal, 2016). Another set of previous works designed hybrid architectures mixing kernel-based methods and ConvNet ideas (Bruna and Mallat, 2013; Huang et al., 2014; Dai et al., 2014; Yang et al., 2015; Oyallon et al., 2017, 2019). Recently, Arora et al. (2020) and Shankar et al. (2020) compared ConvNets to their neural tangent kernel and/or unsupervised kernel network counterparts with a limited set of architectures. We are interested here in whether, given a ConvNet architecture, an analogous CKN can be designed and trained to achieve similar or superior performance. The purely kernel-based approach we adopt here stands in contrast to previous works as, for each

Algorithm 3 Ultimate Layer Reversal Step

Input: Mini-batch of inputs B_t
Overall objective function $f(W, V)$
Current iterates $W^{(t)}$, $V^{(t)}$
Step size γ_t
Regularization parameter τ

Do:

1. Approximate the classifier around the current point $V^{(t)}$:

$$g_{B_t}(V; W^{(t)}) \approx q^{(t)}_{g_{B_t}}(V; W^{(t)}) + \frac{\tau}{2}\|V - V^{(t)}\|_F^2.$$

2. Minimize the approximation:

$$\hat{f}_{B_t}(W^{(t)}) = \min_V q^{(t)}_{g_{B_t}}(V; W^{(t)}) + \frac{\tau}{2}\|V - V^{(t)}\|_F^2.$$

3. Take a projected gradient step:

$$W_\ell^{(t+1)} = \mathrm{Proj}_{S^{d_\ell}}\left(W_\ell^{(t)} - \gamma_t \,\mathrm{Proj}_{S^{d_\ell}}\left(\nabla_{W_\ell}\hat{f}_{B_t}(W^{(t)})\right)\right) \quad \text{for } \ell = 1, \ldots, L.$$

4. Update the classifier on which the approximation is taken:

$$V^{(t+1)} = \operatorname*{argmin}_V q^{(t)}_{g_{B_t}}(V; W^{(t+1)}) + \frac{\tau}{2}\|V - V^{(t)}\|_F^2.$$

Output: $W^{(t+1)}$, $V^{(t+1)}$

(network, data set) pair, we consider a ConvNet and its supervised CKN counterpart, hence compare them on an equal standing, for varying numbers of filters.

2.5.1 Experimental details

The experiments use the data sets MNIST and CIFAR-10, in addition to a 10-class subset of ImageNet 2012 (LeCun et al., 2001; Krizhevsky and Hinton, 2009; Krizhevsky, 2014). MNIST consists of 60,000 training images and 10,000 test images of handwritten digits numbered 0-9 of size 28×28 pixels. In contrast, CIFAR-10 consists of 50,000 training images and 10,000 test images from 10 classes of objects of size $3 \times 32 \times 32$ pixels. The 10-class subset of ImageNet 2012 consists of 5000 training images, 1000 validation images, and 1000 test images of birds

of varying dimension. These images were randomly sampled from the classes lorikeet through goose in the original ImageNet 2012 training set, and the classes are equally represented.

The raw images are transformed prior to being input into the networks. Specifically, the MNIST images are standardized while the CIFAR-10 images are standardized channel-wise and then ZCA whitened on a per-image basis. The ImageNet training images were randomly cropped to size 224×224 pixels, randomly horizontally flipped, and standardized channel-wise. The validation and test set images were scaled so the largest spatial dimension was 256 and then center cropped to 224×224 pixels and standardized channel-wise. Validation sets are created for MNIST and CIFAR-10 by randomly separating the training set into two parts such that the validation set has 10,000 images.

The networks we consider in the experiments are LeNet-1 and LeNet-5 on MNIST (LeCun et al., 2001), All-CNN-C on CIFAR-10 (Springenberg et al., 2015; Krizhevsky and Hinton, 2009), and AlexNet on the subset of ImageNet (Krizhevsky et al., 2012; Krizhevsky, 2014). LeNet-1 and LeNet-5 are prominent examples of first modern versions of ConvNets. They use convolutional layers and pooling/subsampling layers and achieved state-of-the-art performance on digit classification tasks on data sets such as MNIST. The ConvNets from Springenberg et al. (2015), including All-CNN-C, were the first models used to make the claim that pooling is unnecessary. All-CNN-C was one of the best-performing models on CIFAR-10 at the time of publication. Finally, AlexNet revolutionized the field of deep learning by drastically outperforming its competitors in the ImageNet 2012 competition. For mathematical descriptions of the ConvNets and their CKN counterparts, see Appendices A.1 and A.3, respectively.

Using the principles outlined in Section 2.3, we transform each architecture to its CKN counterpart. The networks are in general reproduced as faithfully as possible. However, there are a few differences between the original implementations and ours. In particular, the original LeNets have an incomplete connection scheme at the third layer in which each feature map is only connected to a subset of the feature maps from the previous layer. This was primarily included for computational reasons. In our implementation of the LeNets we

find that converting the incomplete connection scheme to a complete connection scheme does not lead to a large change in the performance (See Appendix A.2). We therefore use the complete connection schemes in our ConvNet and CKN implementations. In addition, the original All-CNN-C has a global average pooling layer as the last layer. In order to have trainable unconstrained parameters in the CKN, we add a fully connected layer after the global average pooling layer in the ConvNet and CKN. Also note that we apply zero-padding at the convolutional layers that have a stride of one to maintain the spatial dimensions at those layers. Moreover, we omit the dropout layers. For the AlexNet ConvNet and CKN we omit the local response normalization, as later work found it had little effect (Simonyan and Zisserman, 2015; Krähenbühl et al., 2016). Lastly, as the arc-cosine kernels are not differentiable, we switch to using the RBF kernel on the sphere for the supervised CKN implementations. The nonlinearity generated by this kernel resembles the ReLU (Mairal et al., 2014b). We fix the bandwidths to 0.6.[4]

The training of the ConvNets used in the experiments is performed as follows. The initialization is performed using draws from a mean-zero random normal distribution. For the LeNets the standard deviation is set to 0.2 while for All-CNN-C the standard deviation is set to 0.3 and for AlexNet the standard deviation is set to 0.1. The output features are normalized in the same way as for the CKNs, so they are centered and on average have an ℓ_2 norm of one. The multinomial logistic loss is used and the network is trained for 10,000 iterations with our ultimate layer reversal method (ULR-SGO). At each ULR-SGO step the full Hessian is approximated. When evaluating the performance of the network, the optimization of the classifier parameters is performed for 1000 iterations. The batch size is set to the largest power of two that fits on the GPU when training the CKN counterpart (see Table A.2 in Appendix A.9). The results we report come from training with ten different random seeds with the hyperparameters fixed to those found during the validation stage (described below). The reported results are from training on the training set rather than the

[4]Note, however, that it is possible to train the bandwidths; see Proposition 34 in Appendix A.4.

combined training and validation sets.

Hold-out validation is performed for several parameters of the ConvNets, namely, the step size, regularization parameter τ of the Hessian, and L_2 penalties on the network and classifier weights. The validation of the L_2 penalty on the classifier parameters is performed over the values 2^i for $i = -40, -39, \ldots, 0$ when fixing the network parameters at their values upon initialization. The step size and parameter τ are initially jointly determined by training for 2000 iterations using the values 2^i for $i = -10, -9, \ldots, -2$ and 2^i for $i = -7, -6, \ldots, -2$, respectively, with the L_2 penalty on the network parameters fixed at zero. Afterward, τ is fixed and the hold-out validation is performed again for 2000 iterations on the step size parameter and the penalty on the network parameters over 2^i for $i = -10, -9, \ldots, -2$. These parameters are then re-validated every 2000 iterations, with the subsequent values selected from $2^i s$ for $i = -3, -2, -1, 0, 1$ for the step size and $2^i \lambda$ for $i = -1, 0, 1, 2, 3$ for the penalty on the network parameters, where s is the current step size and λ is the current penalty. The optimization is performed for 10,000 iterations total for a single random seed.

Now we detail the unsupervised CKN initialization. The unsupervised training of the CKNs entails approximating the kernel at each layer and then training a classifier on top. The kernel approximations are performed using spherical k-means layer-wise with 10,000 randomly sampled non-constant patches per layer, all from different images. Unless otherwise specified, when evaluating the CKN at each layer the intertwined Newton method is used. In order to achieve a high accuracy but keep the computational costs reasonable the number of outer Newton iterations is set to 20 and the number of inner Newton iterations is set to 1. The regularization of the Gram matrix on the filters is set to 0.001. After the unsupervised training the features are normalized as done by Mairal et al. (2014b) so that they are centered and on average have an ℓ_2 norm of one. A classifier is trained on these CKN features using the multinomial logistic loss. The loss function is optimized using L-BFGS (Liu and Nocedal, 1989) on all of the features with the default parameters from the Scipy implementation. Hold-out validation is performed over the values 2^i for $i = -40, -39, \ldots, 0$ for the L_2 penalty of the multinomial logistic loss parameters. Both the hold-out validation and the final

optimization with the optimal penalty are performed for a maximum of 1000 iterations.

Finally, we describe the supervised CKN training. The supervised training of CKNs begins with the unsupervised initialization. The multinomial logistic loss is used and the network is trained with our ultimate layer reversal method. In the ultimate layer reversal method we use an approximation of the full Hessian for all but the LeNet-1 experiment with 128 filters per layer. Due to memory constraints we use a diagonal approximation to the Hessian for the LeNet-1 experiment with 128 filters per layer. The batch size is set to the largest power of two that fits on the GPU (see Table A.2 in Appendix A.9). The hold-out validation for the step sizes and τ are performed in the same way as for the ConvNets.

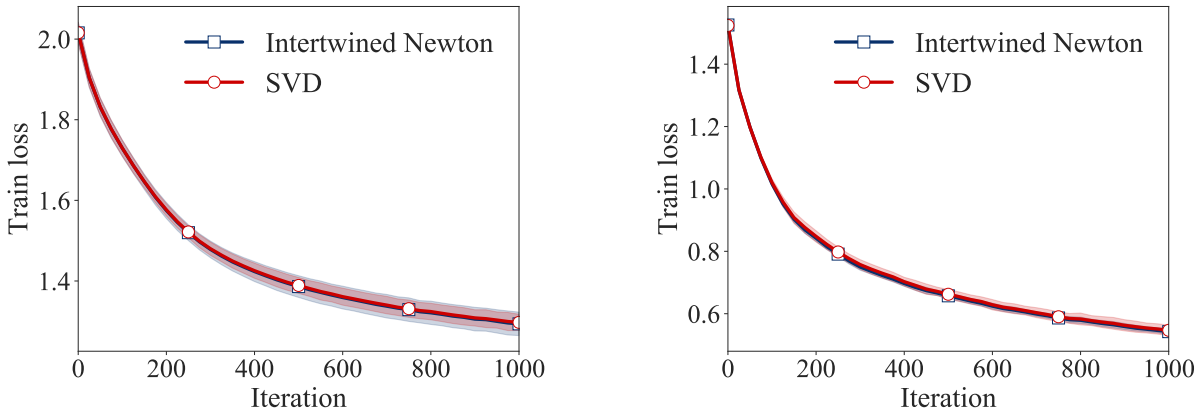
The code for this project was primarily written using PyTorch (Paszke et al., 2019) and may be found online at `https://github.com/cjones6/yesweckn`. FAISS (Johnson et al., 2019) is used during the unsupervised initialization of the CKNs. We ran the experiments on GeForce 1080Tis, Titan Xps, and Titan Vs. The corresponding time to run all of the experiments on an NVIDIA Titan Xp GPU would be more than 20 days.

2.5.2 Comparison of training methods

We commence by demonstrating the superiority of our proposed training methods described in Section 2.4 to the standard methods.

Accuracy of the gradient computation

The straightforward way of computing the gradient of a CKN is by allowing automatic differentiation software to differentiate through SVDs. In Section 2.4.2 we introduced an alternative approach: the intertwined Newton method. Here we compare the two approaches when training the deepest network we consider in the experiments: the CKN counterpart to All-CNN-C. We compare the gradients from differentiating through the SVD and the intertwined Newton method in two ways: directly and also indirectly via the performance when training a CKN.



(a) All-CNN-C CKN on CIFAR-10 with 8 filters/layer

(b) All-CNN-C CKN on CIFAR-10 with 128 filters/layer

Figure 2.2: Average training loss of CKNs when using an SVD to compute the matrix inverse square root vs. using our intertwined Newton method. We report results from using 50 iterations of the outer Newton method and one iteration of the inner Newton method. The error bands represent one standard deviation across 10 trials with different random seeds.

First, we compare the gradients from each method to the result from using a finite difference method. We find that for the CKN counterpart to All-CNN-C on CIFAR-10 with 8 filters/layer, differentiating through 20 Newton iterations yields relative errors that are 2.5 times smaller than those from differentiating through the SVD. This supports the hypothesis that differentiating through the SVD is more numerically unstable than differentiating through Newton iterations. We moreover note that using Newton iterations allows us to control the numerical accuracy of the gradient and of the matrix inverse square root itself.

Given that the gradients from the intertwined Newton method are more accurate, we now investigate whether this makes a difference in the training. Figure 2.2 compares the performance of the two methods on All-CNN-C with 8 and 128 filters/layer. We set the number of outer Newton iterations to 50 and leave the number of inner Newton iterations at 1. From the plots we can see that there is not a large difference in the resultant objective values. For 8 filters/layer the intertwined Newton method achieves an objective value that is

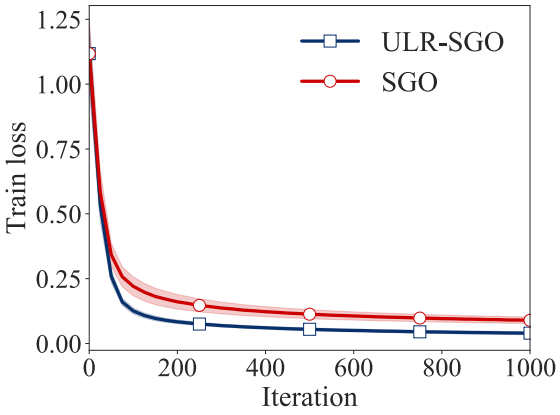
0.3% lower, on average, than the SVD after 1000 iterations. For 128 filters/layer it achieves an objective value that is 0.7% lower, on average, than the SVD after 1000 iterations. This suggests that the effect may be more pronounced for larger networks.

Efficiency of training methods

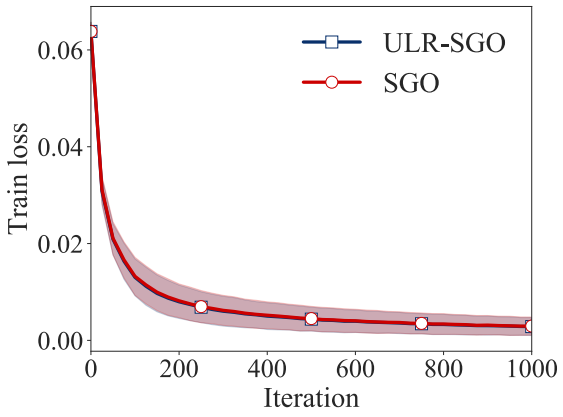
Next, we compare training CKNs using stochastic gradient optimization (SGO) to using our proposed ultimate layer reversal method (ULR-SGO) as detailed in Section 2.4.3. In our SGO implementation we use the version of the optimization in which λ is a penalty parameter rather than a constraint.

Figure 2.3 displays the results of the comparison for the CKN counterparts to LeNet-5 on MNIST and All-CNN-C on CIFAR-10 with 8 and 128 filters/layer. From the plots we can see that ULR-SGO is usually better than SGO throughout the iterations. This difference is most pronounced for the experiments consisting of a more difficult task: classifying CIFAR-10 images with the All-CNN-C CKN. The final accuracy from the ULR-SGO method after 1000 iterations ranges from being nearly the same as that of SGO on the easier task of classifying MNIST digits with the LeNet-5 CKN with 128 filters/layer to 18% better on the harder task of classifying CIFAR-10 images with the All-CNN-C CKN architecture with 8 filters/layer.

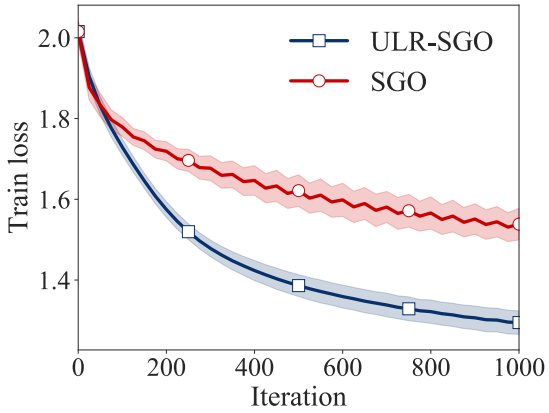
While it is clear that ULR-SGO dominates SGO in terms of its performance across iterations on the harder tasks, it is also important to ensure that this is true in terms of time. Figure A.7 in Appendix A.9 provides the same plots as Figure 2.3, except that the x-axis is now time. The experiments with 8 filters/layer were performed with Nvidia Titan Xp GPUs, while the experiments with 128 filters/layer were performed with Nvidia GeForce 1080Ti GPUs. From the plots we can see that with the exception of the LeNet-5 experiment with 128 filters/layer, the ULR-SGO method still outperforms the SGO method in terms of the training loss vs. time.



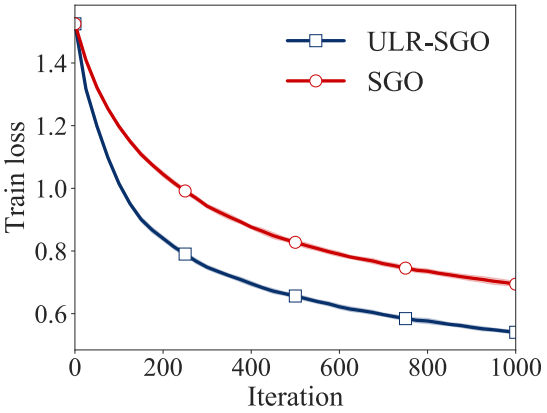
(a) LeNet-5 CKN on MNIST with 8 filters/layer



(b) LeNet-5 CKN on MNIST with 128 filters/layer



(c) All-CNN-C CKN on CIFAR-10 with 8 filters/layer



(d) All-CNN-C CKN on CIFAR-10 with 128 filters/layer

Figure 2.3: Average training loss of CKNs when using our Ultimate Layer Reversal method (ULR-SGO) vs. stochastic gradient optimization (SGO). The error bands represent one standard deviation across 10 trials with different random seeds.

2.5.3 CKNs vs. ConvNets

Now we turn to the comparison between CKNs and ConvNets. We perform this comparison for LeNet-1 and LeNet-5 on MNIST, All-CNN-C on CIFAR-10, and AlexNet on our subset of

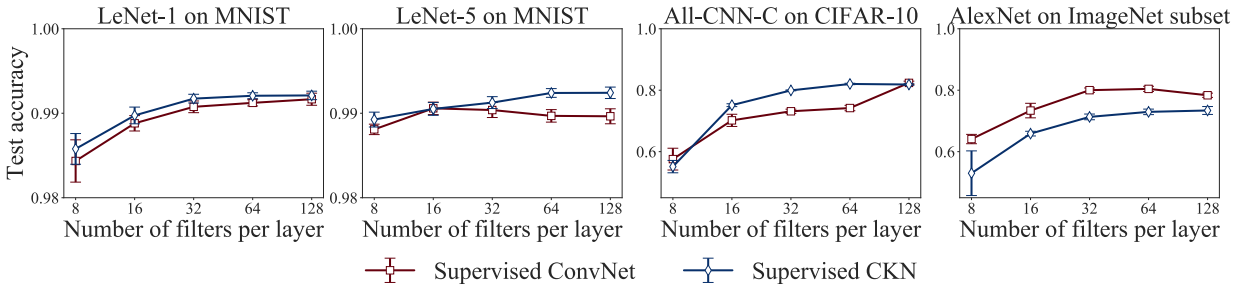


Figure 2.4: Average performance of ConvNets and their CKN counterparts across 10 trials when varying the number of filters per layer. Note that the y-axis scales differ across the plots. The error bars show one standard deviation from the mean.

ImageNet. Figure 2.4 displays the results when we vary the number of filters per layer by powers of two, from 8 to 128. The results are displayed in Figure 2.4 and the corresponding numerical values may be found in Appendix A.9.

Beginning with the LeNets, we see that both the CKN and ConvNet perform well on MNIST over a wide range of the number of filters per layer. The CKN outperforms the ConvNet for almost every number of filters per layer. At best the performance of a CKN is 0.3% better on average and at worst it is approximately the same. The former value is large, given that the accuracies of both the CKNs and the ConvNets exceed 98%. The success of the CKNs continues for All-CNN-C on CIFAR-10. For All-CNN-C the best CKN performance is 11% better on average than that of the ConvNet (in the case of 64 filters/layer) and at worst is 4% worse (in the case of 8 filters/layer). On the other hand, the CKNs do not perform as well as ConvNets for AlexNet on a subset of ImageNet. Indeed, the performance ranges from 6-17% worse. In Figure A.9 in Appendix A.9 we explore whether this could be due to the choice of the kernel. After training AlexNet CKNs with Matérn kernels we find that this may not be the case, as these CKNs perform on par with the ones that use the Gaussian RBF kernels.

Recalling from Section 2.4.3 that CKNs are initialized in an unsupervised manner, we also compare the performance of unsupervised CKNs to their supervised CKN and ConvNet

counterparts. We explore this in Figure A.8 in Appendix A.9. For LeNet-1 the unsupervised CKN performs extremely well, achieving at minimum 98% of the accuracy of the corresponding supervised CKN. The performance is slightly worse for LeNet-5, with the unsupervised CKN achieving 64-97% of the performance of the supervised CKN with the same number of filters. The relative performance is the worst for All-CNN-C, with the unsupervised performance being 44-59% of that of the supervised CKN with the same number of filters. Therefore, the supervised training contributes tremendously to the overall performance of the LeNet-5 CKN with a small number of filters and to the All-CNN-C CKN. These results also suggest that for more complex tasks the unsupervised CKN may require more than 16 times as many filters to achieve comparable performance to the supervised CKN and ConvNet.

2.6 Conclusion

In this work we presented a systematic study of the correspondence between a ConvNet and its CKN counterpart. We also proposed a stochastic gradient algorithm to train CKNs and ConvNets in a supervised manner. When trained using this method, the CKNs we studied achieved comparable performance to their ConvNet counterparts. This study and its results suggests that neural networks and kernel-based methods are two complementary viewpoints to tackle the design of a learning architecture capturing complex invariances, rather than two antagonistic families of approaches as they may appear at the superficial level. We did not address the question of the design of an architecture for a particular task. This question would be interesting to explore in future work.

Chapter 3

REPRESENTATION LEARNING FOR CLUSTER ANALYSIS

Joint work with V. Roulet and Z. Harchaoui.[1]

Abstract. We present an approach for end-to-end learning that allows one to jointly learn a feature representation from unlabeled data (with or without labeled data) and predict labels for unlabeled data. The feature representation is assumed to be specified in a differentiable programming framework, that is, as a parameterized mapping amenable to automatic differentiation. The proposed approach can be used with any amount of labeled and unlabeled data, gracefully adjusting to the amount of supervision. We provide experimental results illustrating the effectiveness of the approach.

3.1 Introduction

Deep networks trained end-to-end are now ubiquitous, being used for tasks ranging from lung cancer screening to music transcription to pose estimation (Ardila et al., 2019; Thickstun et al., 2018; Li et al., 2018). One frequent obstacle when considering new application domains is the need for these networks to be trained in a supervised manner on vast quantities of labeled data. Unfortunately, in many application domains only a small amount of labeled data can be collected or even exists. However, there often exist vast quantities of unlabeled data that are left untouched with a supervised training approach. Recent work surveyed by Oliver et al. (2018) exploits this unlabeled data in a variety of ways to learn feature representations, either with unsupervised or semi-supervised approaches.

[1]This work was first presented at the Women in Machine Learning Workshop in December 2019, for which the first author received travel funding from NSF IIS-1833154. We also gratefully acknowledge support from NSF CCF-1740551, NSF DMS-1810975, the program "Learning in Machines and Brains" of CIFAR, and faculty research awards.

Recently a number of papers have focused on *ad hoc* approaches using unlabeled data in order to achieve the best possible performance in a common domain-specific task. Examples of such approaches include Deep Clustering (Caron et al., 2018) and TagLM (Peters et al., 2017), which achieved remarkable results in computer vision and in natural language processing, respectively. The impressive results obtained on common domain-specific tasks are often achieved thanks to a combination of interesting algorithms and clever tricks informed by expert domain knowledge. These, with the added recourse to industry-scale computing for parameter exploration, result in a lack of clarity of the overall objective actually considered and minimized. Furthermore, the connections with unsupervised learning objectives on the one hand and with supervised learning objectives on the other hand are often partial and unclear.

In this chapter we propose a unified framework for end-to-end learning, applicable when there is only unlabeled data, or some unlabeled data and some labeled data, or only labeled data. The proposed framework motivates a precise objective function to be minimized. Furthermore the connections with classical unsupervised learning and supervised learning objectives are clear. Indeed, the objective naturally reduces to that of a clustering problem when we have no training set labels and that of a classification problem when we have all of the training set labels. Moreover, due to its simplicity, this setup can be extended. For example, indirect constraints on the labels, such as requiring two unlabeled observations to have different labels, can be readily incorporated.

After reviewing related work on unsupervised and semi-supervised learning in Section 3.2, we present the framework in Section 3.3. We then focus on a specific objective in Section 3.4, showing that our proposed objective is smoother than a straightforward alternative. We address how to optimize the objective function in Section 3.4.2. Optimizing over the labels requires care, and for this we present a novel algorithm based on a convex relaxation of the problem. Finally, we demonstrate the proposed approach in Section 3.5, showing that our method, called XSDC, outperforms the supervised baseline.

3.2 Related Work

In this work we propose a method for clustering that (1) works with any ratio of labeled to unlabeled data; and (2) learns feature representations. In this section we first survey prior approaches to clustering that work with varying levels of supervision. We then describe recent approaches to learning feature representations when no labeled data is available and when some labeled data is available.

Semi-supervised clustering. Intuitively, there are two main ways of developing a clustering algorithm that can work with both labeled and unlabeled data. First, one could modify a supervised classification method so that it can incorporate unlabeled data. Such modifications come in different flavors, including adding a penalty to a supervised learning objective to encourage similar inputs to be close together in feature space (Belkin et al., 2006; Bachman et al., 2014; Kamnitsas et al., 2018; Iscen et al., 2019), adding a penalty to encourage high-confidence outputs (Grandvalet and Bengio, 2004), or rounding outputs to obtain pseudo-labels (Lee, 2013; Berthelot et al., 2019). Other approaches add a supervised loss to an unsupervised loss (Beyer et al., 2019). Alternatively, one could modify a clustering algorithm in order to incorporate labeled data. Approaches of this kind include constrained "reverse prediction" based on a k-means formulation and generalizations thereof (Xu et al., 2009; White and Schuurmans, 2012). See the surveys of Chapelle et al. (2010) and Oliver et al. (2018) for a broader overview of semi-supervised algorithms.

The approach we take is based on DIFFRAC (Bach and Harchaoui, 2007), which falls in the former class of methods. DIFFRAC may be thought of as ridge regression where the output matrix Y is a one-hot cluster assignment matrix and where one optimizes over both Y and the classifier parameters. In order to avoid trivial solutions, cluster size constraints are enforced. Various extensions of DIFFRAC have also been considered in the literature (Joulin and Bach, 2012; Flammarion et al., 2017). The advantage of the objective introduced by Bach and Harchaoui (2007) is that it allows one to easily incorporate additional information

about the clustering problem. Namely, it paved the way to several popular weakly supervised techniques developed by Bojanowski et al. (2014, 2015) and Alayrac et al. (2016) for computer vision problems.

Representation learning. A large number of representation learning methods exist. Here we survey representation learning methods that are unsupervised and work with only unlabeled data or that are semi-supervised and work with both labeled and unlabeled data.

Most unsupervised deep feature learning methods can be broadly classified into one of two categories: methods that optimize a surrogate loss, often based on known structure in the data; and methods that directly optimize a loss function of interest. Early examples of the former set of methods include auto-encoders, which attempt to reconstruct the input observations through a deep network (LeCun, 1987; Goodfellow et al., 2016). Other more recent examples attempt to approximate a kernel at each layer of a network (Bo et al., 2011; Mairal et al., 2014b; Daniely et al., 2017). Most recently, many papers have been taking advantage of structure in the data. This includes training to distinguish between multiple views of images or patches and other images or patches (Wang and Gupta, 2015; Dosovitskiy et al., 2016; Sermanet et al., 2018; Bachman et al., 2019), learning to predict the relative location of patches in images (Doersch et al., 2015; Noroozi and Favaro, 2016), and predicting color from grayscale images (Zhang et al., 2016). It also includes learning to distinguish segments within time series or patches within images, or to predict future observations in time series (Hyvärinen and Morioka, 2016; van den Oord et al., 2018; Löwe et al., 2019). A downside to these latter approaches is the focus on achieving state-of-the-art results on domain-specific tasks in computer vision and signal processing at the expense of the conciseness of the formulation.

The second category of unsupervised methods typically alternately optimizes the parameters of the network and the labels or cluster assignments of the observations. In this thread, several papers alternate between obtaining assignments or soft assignments and optimizing the parameters of a loss function aimed at creating well-separated clusters (Xie

et al., 2016; Yang et al., 2016; Ghasedi Dizaji et al., 2017; Häusser et al., 2017). In contrast, Bojanowski and Joulin (2017) randomly generate outputs and then alternately optimize over the parameters of the model and the assignment of labels to outputs. The most direct approach may be that of Caron et al. (2018), who alternately cluster the data to obtain pseudo-labels and take steps to optimize the multinomial logistic loss on the observations with the given pseudo-labels. A drawback of these approaches is the design of an *ad-hoc* objective not clearly related to objectives commonly used in unsupervised clustering or supervised classification, or the combined use of two different objectives, one for optimizing the network and one for clustering.

The category of semi-supervised representation learning methods includes a number of the semi-supervised clustering methods discussed above. Lee (2013); Kamnitsas et al. (2018); Berthelot et al. (2019); Beyer et al. (2019), and Iscen et al. (2019) all propose methods for learning features in the presence of unlabeled data. This category also includes approaches that learn a feature representation in an unsupervised manner before fine-tuning with labeled data (e.g., Wu et al., 2018). The downside to these approaches is that they either do not use a single objective function or they are not designed to work in the purely unsupervised setting. In this chapter we build our formulation on a unified objective that encompasses learning with unlabeled data only, learning with labeled and unlabeled data, and learning with labeled data only.

Relation to existing methods. This work may be viewed as an extension of DIFFRAC (Bach and Harchaoui, 2007) for learning feature representations. As argued in Chapter 2, a feature representation defined by a deep network can be related to an approximation of a feature map associated with a composition of kernels. From this viewpoint, the approach in this chapter can also be interpreted as learning a kernel, i.e., a similarity measure, acting on pairs of examples. Learning a similarity measure for clustering was previously explored by Meila et al. (2005) in the supervised setting.

In addition to using deep networks, we improve upon the work of Bach and Harchaoui

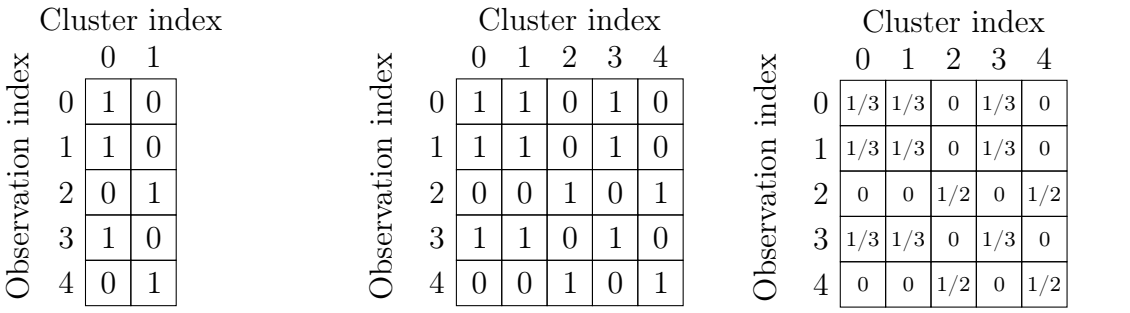
(2007) by proposing a simplified convex relaxation of the labeling subproblem. This relaxation allows us to handle several types of constraints on the labels. It is similar to the problem Zass and Shashua (2006) solved to find a doubly stochastic matrix for use in spectral clustering. However, the problem we consider is more general and we add an additional regularization term. This regularization term makes the problem strictly convex and enforces non-negativity of the minimizer. The labeling procedure we propose recovers the Sinkhorn-Knopp algorithm (Sinkhorn and Knopp, 1967; Peyré and Cuturi, 2019) when there is no labeled data and the sizes of the clusters are assumed to be known. Since this chapter was first submitted, Asano et al. (2020) explored a similar approach aimed at representation learning for unsupervised cluster analysis, with a focus on computer vision problems such as image classification and object detection. In contrast to our approach, their method is based on a batch optimization algorithm. Moreover, their formulation is parameterized with respect to the label (assignment) matrix rather than the equivalence matrix.

3.3 Learning with any Level of Supervision

Any clustering method based on optimizing an objective function can potentially be transformed into a method that works with any level of supervision. Often such clustering methods, including k-means, various forms of spectral clustering, and DIFFRAC, are viewed as independent algorithms, ignoring their connections. In this section we first describe a framework for clustering and show how each of these algorithms is embedded as a special case. We then discuss how this framework can be adapted in order to use any amount of labeled data.

3.3.1 The M-family of clustering methods

Consider observations $x_1, \ldots, x_n \in \mathbb{R}^d$, each belonging to one of k (unknown) clusters. There are numerous ways to represent the assignments of the observations $x_1, \ldots, x_n$ to clusters (see, e.g., Zha et al., 2001; Bach and Jordan, 2006; Roulet, 2017). First, one can use the assignment matrix $Y \in \{0,1\}^{n \times k}$, where $Y_{i,\cdot} =: y_i$ is a one-hot cluster assignment vector for observation i. Alternatively, one can use the equivalence matrix $M = YY^T$. In this matrix,



(a) Assigment matrix Y

(b) Equiv. matrix $M = YY^T$

(c) Normalized equiv. matrix $\tilde{M} = YY^T(YY^T)^\dagger$

Figure 3.1: Three ways to represent clusterings.

entry (i, j) is 1 if observations i and j belong to the same cluster and is 0 otherwise. Finally, one can use the normalized equivalence matrix $\tilde{M} = YY^T(YY^T)^\dagger$, where $M^\dagger$ denotes the pseudo-inverse of the matrix M. In this matrix, entry (i, j) is $1/n_i$ if observations i and j belong to the same cluster and is 0 otherwise, where n_i is the number of elements in the cluster observations i and j belong to. An example of each of these representations is depicted in Figure 3.1.

Now let $S \in \mathbb{R}^{n \times n}$ be a given similarity matrix derived from the observation matrix $X = [x_1, \ldots, x_n]^T$, where S_{ij} is the similarity between observations i and j. Consider the problem of assigning observations to clusters. Intuitively, we want to maximize the similarity of points within each cluster. In other words, we might consider solving $\max_{\tilde{M}} \langle S, \tilde{M} \rangle_F$ or $\max_M \langle S, M \rangle_F$ subject to the constraint that each observation lies in exactly one cluster and $\tilde{M}$ or M is a (normalized) equivalence matrix. In each case trivial solutions can exist (e.g., in the second case, if S is strictly positive, then a trivial solution assigns all observations to the same cluster). To avoid such solutions we can add constraints on the cluster sizes. As we will see shortly, for particular choices of S, these two problems can lead to previously-established clustering algorithms, such as k-means and DIFFRAC (MacQueen, 1967; Bach and Harchaoui, 2007). This intuition motivates the following family of clustering problems that we study in

this section. To align with traditional clustering objectives we write the problem in terms of minimization rather than maximization.

Definition 6. *Let $\Psi_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times n}$ and $\Gamma_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times n}$ be functions parameterized by $\theta \in \mathbb{R}^{d_\theta}$ for some d_θ. The M-family of clustering problems is then given by*

$$\begin{aligned} \min_{Y} \quad & \left\langle \Psi_\theta(X), (\Gamma_\theta(X)YY^T\Gamma_\theta(X))^\alpha (\Gamma_\theta(X)YY^T\Gamma_\theta(X))^{\dagger\beta} \right\rangle_F \\ \textit{subject to} \quad & Y\mathbb{1}_k = \mathbb{1}_n \\ & \lambda_1 \left(Y^T\mathbb{1}_n - n_{\min}\mathbb{1}_k\right) \geq 0 \\ & \lambda_2 \left(Y^T\mathbb{1}_n - n_{\max}\mathbb{1}_k\right) \leq 0 \\ & y_{ij} \in \{0,1\} \quad \forall\, i,j\ , \end{aligned} \tag{3.1}$$

where $\alpha, \beta, \lambda_1, \lambda_2 \in \{0,1\}$ and $n_{\min}, n_{\max} > 0$.

We call this family the M-family because the objective can be written exclusively in terms of $M = YY^T$. Since we are minimizing rather than maximizing the objective, we can think of $\Psi_\theta(X)$ as a dissimilarity matrix on the observations parametrized by θ. Usually $\Gamma_\theta(X)$ is the identity. However, for normalized cut spectral clustering (Shi and Malik, 2000), where it is the degree matrix, we can think of $\Gamma_\theta(X)$ as reweighting the entries of M according to the importance of each observation. Next we show how we can recover k-means, DIFFRAC, and normalized cut spectral clustering, given particular choices of $\Psi_\theta, \Gamma_\theta, \alpha, \beta, \lambda_1$, and λ_2. Table 3.1 summarizes how these methods and some other common clustering algorithms fit into the M-family.

Example 2 (k-means)**.** *Define $\Psi_\theta(X) = -XX^T$ and $\Gamma_\theta(X) = \mathrm{I}_n$, and let $\alpha = \beta = 1$ and $\lambda_1 = \lambda_2 = 0$. The resultant M-family problem is given by*

$$\begin{aligned} \min_{Y} \quad & \left\langle -XX^T, YY^T(YY^T)^\dagger \right\rangle_F \\ \textit{subject to} \quad & Y\mathbb{1}_k = \mathbb{1}_n \end{aligned}$$

$$y_{ij} \in \{0,1\} \quad \forall\, i,j\ .$$

Adding a term $\langle X, X\rangle$ to the objective, which does not affect the minimizer, and using the fact that for a matrix Z, $ZZ^\dagger = ZZ^T(ZZ^T)^\dagger$ (Lütkepohl, 1996, p. 35), we obtain

$$\begin{aligned}\left\langle XX^T, \mathrm{I}_n - YY^T(YY^T)^\dagger \right\rangle_F &= \left\langle XX^T, \mathrm{I}_n - YY^\dagger \right\rangle_F \\ &= \|X - YY^\dagger X\|_F^2 \\ &= \min_{\mu \in \mathbb{R}^{k\times d}} \|X - Y\mu\|_F^2\ .\end{aligned}$$

The overall problem can then be written as

$$\begin{aligned}\min_{Y,\mu} \quad & \|X - Y\mu\|_F^2 \\ \textit{subject to} \quad & Y\mathbb{1}_k = \mathbb{1}_n \\ & y_{ij} \in \{0,1\} \quad \forall\, i,j\ ,\end{aligned}$$

which is precisely the k-means problem.

Example 3 (DIFFRAC with cluster size constraints)**.** *Define $\Psi_\theta(X) = A_\lambda(X)$ where $A_\lambda(X) \coloneqq \lambda\Pi_n \left(\Pi_n XX^T\Pi_n + n\lambda\,\mathrm{I}\right)^{-1}\Pi_n$ and $\Pi_n = \mathrm{I}_n - \mathbb{1}_n\mathbb{1}_n^T/n$. Furthermore, define $\Gamma_\theta(X) = \mathrm{I}_n$. Let $\alpha = 1, \beta = 0$, and $\lambda_1 = \lambda_2 = 1$. These choices of the M-family parameters lead to the problem*

$$\begin{aligned}\min_{Y} \quad & \langle A_\lambda(X), YY^T\rangle_F \\ \textit{subject to} \quad & Y\mathbb{1}_k = \mathbb{1}_n \\ & Y^T\mathbb{1}_n \geq n_{\min}\mathbb{1}_k \\ & Y^T\mathbb{1}_n \leq n_{\max}\mathbb{1}_k \\ & y_{ij} \in \{0,1\} \quad \forall\, i,j\ .\end{aligned}$$

This is precisely the DIFFRAC problem of Bach and Harchaoui (2007, equation 2) with cluster size constraints. Bach and Harchaoui (2007) showed that this problem is the same as the following ridge regression problem when also optimizing over Y:

$$\min_{Y \in \mathcal{C}_Y^D, W \in \mathbb{R}^{d \times k}, b \in \mathbb{R}^k} \frac{1}{n} \sum_{i=1}^{n} \|y_i - (W^T x_i + b)\|_2^2 + \lambda \|W\|_F^2 \,, \tag{3.2}$$

where $\mathcal{C}_Y^D = \{Y \in \{0,1\}^{n \times k} : Y \mathbb{1}_k = \mathbb{1}_n, n_{\min} \mathbb{1}_k \leq Y^T \mathbb{1}_n \leq n_{\max} \mathbb{1}_k\}$ is the constraint set on the labels.

Example 4 (Normalized cut spectral clustering)**.** *Let $S_\theta \in \mathbb{R}^{n \times n}$ be a non-negative, symmetric similarity matrix derived from X (e.g., $(S_\theta)_{ij} = \exp(-\|x_i - x_j\|^2/(2\theta^2))$). Given such a matrix S_θ, which we will henceforth denote by S, define the degree matrix $D = \operatorname{diag}([D_i]_{i=1}^n)$ where $D_i = \sum_{j=1}^n S_{ij}$ for all i and the Laplacian matrix $L = D - S$. Assume the degree D_{ii} for each observation i is strictly positive such that the degree matrix D is invertible. Define $\Psi_\theta(X) = D^{-1/2} L D^{-1/2}$ and $\Gamma_\theta(X) = D^{1/2}$, and let $\alpha = \beta = 1$ and $\lambda_1 = \lambda_2 = 0$. These choices lead to the M-family problem given by*

$$\begin{aligned} \min_{Y} \quad & \left\langle D^{-1/2} L D^{-1/2}, (D^{1/2} Y Y^T D^{1/2})(D^{1/2} Y Y^T D^{1/2})^\dagger \right\rangle_F \\ \textit{subject to} \quad & Y \mathbb{1}_k = \mathbb{1}_n \\ & Y^T \mathbb{1}_n \geq \mathbb{1}_k \\ & y_{ij} \in \{0,1\} \quad \forall\, i, j \,. \end{aligned}$$

The additional constraint $Y^T \mathbb{1}_n \geq \mathbb{1}_k$ ensures that there is at least one point per cluster. Without this constraint, the solution to the problem would assign all points to the same cluster. Using the fact that for a matrix Z, $ZZ^\dagger = ZZ^T(ZZ^T)^\dagger$ (Lütkepohl, 1996, p. 35), we can rewrite the objective as

$$\operatorname{trace}\left[D^{-1/2} L D^{-1/2} (D^{1/2} Y Y^T D^{1/2})(D^{1/2} Y Y^T D^{1/2})^\dagger\right]$$

$$
\begin{aligned}
&= \text{trace}\left(D^{-1/2}LD^{-1/2}D^{1/2}Y(D^{1/2}Y)^{\dagger}\right) \\
&= \text{trace}\left((D^{1/2}Y)^{\dagger}D^{-1/2}LY\right) \\
&= \text{trace}\left((Y^TDY)^{-1}Y^TD^{1/2}D^{-1/2}LY\right) \\
&= \text{trace}\left((Y^TDY)^{-1}Y^TLY\right) \\
&= \text{trace}\left(\text{diag}((Y_{\cdot,1}^TDY_{\cdot,1})^{-1},\ldots,(Y_{\cdot,k}^TDY_{\cdot,k})^{-1})Y^TLY\right) \\
&= \sum_{j=1}^{k}\frac{Y_{\cdot,j}^TLY_{\cdot,j}}{Y_{\cdot,j}^TDY_{\cdot,j}} \ .
\end{aligned}
$$

Let $\mathscr{C} = \{C_1, \ldots, C_k\}$ define a clustering, where each C_j is a set containing the indices of the observations in cluster j. Moreover, denote the sum of the degrees of nodes in a set C by Vol(C), the volume of C. Spectral clustering traditionally makes use of the concept of a "cut" between two sets C and C', defined to be the sum of the similarities between elements in set C and in set C':

$$Cut(C, C') \coloneqq \sum_{i\in C}\sum_{j\in C'} S_{ij} \ .$$

With this definition, we may rewrite the objective as

$$\sum_{j=1}^{k}\frac{Y_{\cdot,j}^TLY_{\cdot,j}}{Y_{\cdot,j}^TDY_{\cdot,j}} = \sum_{j=1}^{k}\sum_{j\neq j'}\frac{Cut(C_j, C_{j'})}{Vol(C_j)} \ ,$$

where the last line follows from observing that $Vol(C_j) = Y_{\cdot,j}^TDY_{\cdot,j}$ and $\sum_{j'\neq j} Cut(C_j, C_{j'}) = Y_{\cdot,j}^T(D-S)Y_{\cdot,j}$ (Xing and Jordan, 2003). Therefore, the problem may be written as

$$
\begin{aligned}
\min_{\mathscr{C}=\{C_1,\ldots,C_k\}} \quad & \sum_{j=1}^{k}\sum_{j'\neq j}\frac{Cut(C_j, C_{j'})}{Vol(C_j)} \\
\textit{subject to} \quad & C_j \cap C_{j'} = \emptyset \quad \forall\, j \neq j' \\
& \cup_{j=1}^{k} C_j = \{1, \ldots, n\} \ ,
\end{aligned}
$$

Algorithm	$\Psi_\theta(X)$	$\Gamma_\theta(X)$	α	β	λ_1	λ_2
Correlation clustering (Swamy, 2004)	$(w_{out} - w_{in})^T$	I_n	1	0	0	0
DIFFRAC (Bach and Harchaoui, 2007)	$A_\lambda(X)$	I_n	1	0	1	1
DIFFRAC-cosegmentation (Joulin et al., 2010)	$A_\lambda(X) + \mu/nL(X)$	I_n	1	0	1	1
k-means (MacQueen, 1967)	$-XX^T$	I_n	1	1	0	0
Kernel k-means (Schölkopf et al., 1998)	$-K$	I_n	1	1	0	0
Spectral clustering (Balanced cut) (Wu and Leahy, 1993)	L	I_n	1	0	1	1
Spectral clustering (NCut) (Shi and Malik, 2000)	$D^{-1/2}LD^{-1/2}$	$D^{1/2}$	1	1	1	0
Spectral clustering (Ratio Cut) (Hagen and Kahng, 1992)	L	I_n	1	1	1	0
Stochastic block model* (Jalali et al., 2016)	$-A \log \frac{p_\tau(1-q)}{1-p_\tau q} - \mathrm{I}_n \log \frac{1-p_\tau}{1-q}$	I_n	1	0	0	0

Table 3.1: Examples of clustering methods belonging to the M-family. If not specified in the text, the notations used are the same as those in the references. *The stochastic block model formulation assumes that the p_i's and q are known and $p_{\tau(i,j)} = p_\tau$ for all i, j, i.e., it is a homogeneous stochastic block model.

which is precisely the multi-way normalized cut clustering problem.

When $\Psi_\theta(X)$ and $\Gamma_\theta(X)$ are derived from a version of spectral clustering, the method has a random-walk interpretation, as first described by Meila and Shi (2001). By normalizing the rows of the similarity matrix S to sum to 1 we obtain a matrix $P \coloneqq D^{-1}S$ that can be viewed as a stochastic transition matrix. Moreover, defining $\pi_i = D_i/\mathrm{Vol}(\mathscr{C})$, we can see that $P^T\pi = \pi$ and hence π is a stationary distribution of P. We also define $\pi_C \coloneqq \mathrm{Vol}(C)/\mathrm{Vol}(\mathscr{C})$.

We can interpret the normalized cut criterion as the sum of the probabilities of moving from one cluster to another. To see this, observe that the probability of going from one cluster C to another cluster C' is given by

$$P_{C'|C} = \frac{\sum_{i\in C, j\in C'} \pi_i P_{ij}}{\pi_C} = \frac{\sum_{i\in C, j\in C'} S_{ij}}{\mathrm{Vol}(C)} = \frac{\mathrm{Cut}(C, C')}{\mathrm{Vol}(C)} \,. \tag{3.3}$$

In contrast, when using ratio cut, we obtain a weighted sum of the probabilities of going from one cluster C to a different cluster C', with weights $\mathrm{Vol}(C)/|C|$, i.e., twice the average degree of an element of C. Similarly, when using balanced min-cut, we obtain a weighted sum of the probabilities of going from one cluster to another, with weights $\mathrm{Vol}(C)$.

3.3.2 Incorporation of labeled data

A natural way to take labeled data into account in the M-family objective is to add additional constraints on the labels. Specifically, we now consider the case where the one-hot cluster label for each observation i, $y_i^\star$, may or may not be observed. We denote by $\mathcal{S}$ the set of indices corresponding to the labeled data and by $\mathcal{U}$ the set of indices corresponding to the unlabeled data. In this case the constraint set on the label matrix Y becomes $\mathcal{C}_Y = \{Y \in \{0,1\}^{n\times k} : Y\mathbb{1}_k = \mathbb{1}_n, \lambda_1\left(Y^T\mathbb{1}_n - n_{\min}\mathbb{1}_k\right) \geq 0, \lambda_2\left(Y^T\mathbb{1}_n - n_{\max}\mathbb{1}_k\right) \leq 0, y_i = y_i^\star \text{ for } i \in \mathcal{S}\}$. We can translate this constraint set to the following constraint set on the equivalence matrix M: $\mathcal{C}_M = \{M \in \{0,1\}^{n\times n} : \exists Y \in \mathcal{C}_Y \text{ s.t. } M = YY^T\}$. In addition to optimizing over the entries of M, we can also consider optimizing over the parameters θ.

The advantage of this problem formulation is that it captures three regimes: the unsupervised regime, in which clustering is performed to learn the equivalence matrix M in addition to the parameters θ; the supervised regime, in which supervised training is performed to learn the parameters θ; and the semi-supervised regime, in which a combination of clustering and supervised training are performed to learn the unknown elements of M, in addition to the parameters θ. Given a specific M-family clustering objective and an optimization algorithm, we may therefore proceed with training regardless of the amount of labeled data. Figure 3.2 displays examples of the equivalence matrix M and the problem in the cases of no labeled data, some labeled data, and fully-labeled data.

3.4 Extension of the DIFFRAC Objective

In the remainder of this chapter we focus on the extension of DIFFRAC objective to the case where it can both learn a feature representation and estimate the labels of any quantity of unlabeled observations.

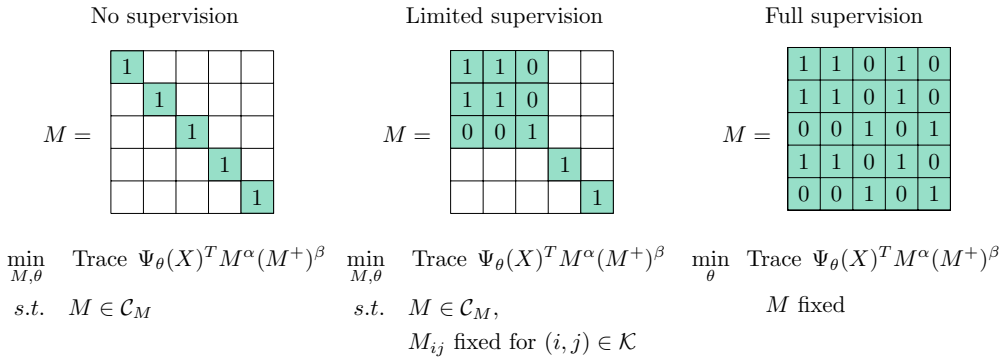


Figure 3.2: Example equivalence matrix M and objective function for varying levels of supervision. For simplicity we set $\Gamma_\theta(X) = \mathrm{I}_n$ in the objective functions.

3.4.1 Problem formulation

In order to learn a feature representation, we propose transforming the input to the original DIFFRAC objective (3.2) using a differentiable deep network $\phi_V : \mathbb{R}^d \to \mathbb{R}^D$. We aim to use both the labeled and unlabeled data to learn (a) the parameters V_ℓ at each layer $\ell = 1, 2 \ldots, m$ of ϕ, where $V = \{V_1, \ldots, V_m\}$; and (b) the parameters $W \in \mathbb{R}^{D \times k}$ and $b \in \mathbb{R}^k$ of the classifier on the output features $\phi_V(x_i)$, $i = 1, \ldots, n$. For simplicity we will assume there exists a constant B such that for all V and for all $x \in \mathbb{R}^d$, $\|\phi_V(x)\|_2 \leq B$, i.e., the network has bounded outputs.

To this end, we consider solving the problem

$$\min_{Y \in \mathcal{C}_Y^D, V, W, b} \frac{1}{n} \sum_{i=1}^{n} \|y_i - (W^T x_i + b)\|_2^2 + \mathcal{R}(V, W) \;, \tag{3.4}$$

where $\mathcal{C}_Y^D = \{Y \in \{0,1\}^{n \times k} : Y \mathbb{1}_k = \mathbb{1}_n, n_{\min} \mathbb{1}_k \leq Y^T \mathbb{1}_n \leq n_{\max} \mathbb{1}_k, y_i = y_i^\star \text{ for } i \in \mathcal{S}\}$ is the constraint set on the labels and $\mathcal{R}(V, W) \coloneqq \alpha \sum_{j=1}^m \|V_j\|_F^2 + \lambda \|W\|_F^2 - \rho \sum_{i=1}^n \|\phi_V(x_i) - \bar{\phi}\|_2^2$ contains the regularization terms. Here $\alpha \geq 0$, $\lambda \geq 0$, and $\rho \geq 0$ are regularization parameters. We have added an additional regularization on the network parameters V_j to promote smoothness of the learned network. We have also added a penalty on the trace of the

covariance matrix of the features to encourage the learned features for each observation to be different. In the following we denote simply $\phi_i(V) = \phi_V(x_i)$ and $\Phi(V) = (\phi_1(V), \ldots, \phi_n(V))^T$.

We shall in Section 3.4.2 present an algorithm to optimize this objective. We term the overall algorithm *XSDC* for "X-Supervised Discriminative Clustering", where "X" can be "un", "semi" or "-", hence covering all cases.

Comparison to reverse prediction objective. The main component of our objective is regularized "forward prediction" least squares, which is given by $\|Y - \Phi(V)W - \mathbb{1}_n b^T\|_F^2/n + \lambda\|W\|_F^2$ for features $\Phi(V)$. We could alternatively consider "reverse prediction" least squares, which is given by $\|\Phi(V) - YW\|_F^2/n$ and is used in k-means. In both cases we would alternate between updating the parameters V and W and estimating the labels Y.

One way in which we can compare the quality of the objectives generated by these two options for learning a representation is via their smoothness properties, i.e., their Lipschitz continuity and the Lipschitz continuity of their gradients. These control the step sizes of optimization methods; see Bertsekas (2016) and Nesterov (2018) for a discussion of the interplay between smoothness properties and rates of convergence. We now proceed to show that when fixing the labels Y the forward prediction objective is smoother than the reverse prediction objective for appropriate choices of the regularization parameter λ.

For both objectives, we consider fixed labels $Y \in \{0,1\}^{n\times k}$ with $Y\mathbb{1}_k = \mathbb{1}_n$. Moreover, for simplicity we will take $\alpha = \rho = 0$. Consider the "forward prediction" objective from (3.4). Define the centering matrix $\Pi_n = \mathrm{I}_n - \mathbb{1}_n\mathbb{1}_n^T/n$. After minimizing over the bias b, the problem may be written as

$$\begin{aligned}\min_V F_f(\Phi(V)) &:= \min_{V,W} \frac{1}{n}\|\Pi_n[Y - \Phi(V)W]\|_F^2 + \lambda\|W\|_F^2 \\ &= \min_V \operatorname{trace}[YY^T A_\lambda(\Phi(V))]\ , \end{aligned} \tag{3.5}$$

where $A_\lambda(\Phi) = \lambda\Pi_n\left(\Pi_n\Phi\Phi^T\Pi_n + n\lambda\,\mathrm{I}_n\right)^{-1}\Pi_n$.

The corresponding "reverse prediction" problem is given by

$$\begin{aligned}\min_V F_r(\Phi(V)) &\coloneqq \min_{V,W} \frac{1}{n}\|\Phi(V) - YW\|_F^2 \\ &= \min_V \frac{1}{n}\operatorname{trace}[(\mathrm{I} - P_Y)\Phi(V)\Phi(V)^T] \,,\end{aligned}$$

where $P_Y = Y(Y^TY)^{-1}Y^T$ is an orthonormal projector.

To compare the smoothness with respect to any matrix V_j, $j = 1, \ldots, m$ it suffices to compute the smoothness with respect to Φ. The next two propositions do that and suggest that our "forward prediction" objective is generally smoother than the "backward prediction" objective. The proofs may be found in Appendix B.1.

Proposition 7. *Let $\mathcal{Z}$ be the set of all possible feature matrices $\Phi \in \mathbb{R}^{n\times D}$. Assume there exists $B \in \mathbb{R}$ such that for all $\Phi \in \mathcal{Z}$, $\|\Phi\|_2 \leq B$. Let $n_{\max}$ be a bound on the maximum number of points in a cluster. Then the Lipschitz constants of F_f and F_r with respect to the spectral norm can be estimated by*

$$L_f \coloneqq 2B\left(\frac{n_{\max}}{n}\right)\left(\frac{1}{n\lambda}\right) \quad \text{and} \quad L_r \coloneqq \frac{2}{n}B \,,$$

respectively. We therefore have $L_f \leq L_r$ for $\lambda \geq n_{\max}/n$.

Proposition 8. *Under the same assumption as Proposition 7, the Lipschitz constants of ∇F_f and ∇F_r with respect to the spectral norm can be estimated by*

$$\ell_f \coloneqq 2\left(\frac{n_{\max}}{n}\right)\left(\frac{1}{n\lambda}\right) + 8B^2\left(\frac{n_{\max}}{n}\right)\left(\frac{1}{n^2\lambda^2}\right) \quad \text{and} \quad \ell_r \coloneqq \frac{2}{n} \,,$$

respectively. We therefore have $\ell_f \leq \ell_r$ for $\lambda \geq n_{\max}/(2n) + \sqrt{n_{\max}^2 + 16B^2 n_{\max}}/(2n)$.

3.4.2 Optimization

The algorithm XSDC that we propose to optimize the objective function (3.4) works on mini-batches. Below we will see that, with the square loss, we only ever need to work with

the equivalence matrix $M \coloneqq YY^T$ rather than the label matrix Y itself during training. Therefore, at each iteration we first optimize M for a mini-batch given fixed V, W, and b. Then we update V, W, and b for fixed M. Optimizing over M is the difficult part, and we propose a novel method for doing so. For the optimization over V, W, and b we use the Ultimate Layer Reversal Stochastic Gradient Optimization method (ULR-SGO) from Chapter 2, which we review next.

Ultimate layer reversal. Rather than using stochastic gradient optimization to learn V, W, and b, we use the ULR-SGO method from Chapter 2. It proceeds as follows. Denote the objective function (3.4) for fixed Y by $F_{\text{ulr}}(V, W, b)$. At each iteration, compute $\hat{F}_{\text{ulr}}(V) \coloneqq \min_{W,b} F_{\text{ulr}}(V, W, b)$, rewriting the objective exclusively in terms of V. Then, using $\hat{F}_{\text{ulr}}$, update V by taking one gradient step.

As long as F_{ulr} is twice differentiable and F_{ulr} viewed as a function of W and b is strongly convex, gradient descent on this objective converges to a stationary point and the resultant $\varepsilon-$ stationary points are $\varepsilon-$ stationary points of the original problem. If $\hat{F}_{\text{ulr}}(V)$ is not available in closed form we may estimate it using a quadratic approximation of the loss around the current estimate of V. In addition, this method can also be applied on mini-batches and in the setting where V is constrained.

Optimizing using ULR-SGO has two main benefits. First, it was empirically shown in Chapter 2 to converge faster than standard stochastic gradient optimization. Second, in the case of the square loss it allows us to work with the equivalence matrix $M = YY^T$ rather than the assignment matrix Y during the alternating optimization. To see this, observe that from Equation (3.5) we have

$$\hat{F}_{\text{ulr}}(V) = \text{trace}[MA_\lambda(\Phi(V))] + R(V) \ ,$$

where A_λ is defined as in Equation (3.5) and the regularization term is $R(V) \coloneqq \alpha \sum_{j=1}^{m} \|V_j\|_F^2 - \rho \|\Pi_n \Phi(V)\|_F^2$. Since we only need to optimize over M we can avoid dealing with the problem

of there being many solutions $Y^\star$ caused by the optimal objective value being the same if the columns of Y are permuted. In practice this means that we avoid performing an additional rounding step to obtain the assignment matrix.

Matrix balancing. Next, consider the objective function (3.4) when fixing V, W, and b and optimizing over only the equivalence matrix $M = YY^T$. As shown in Proposition 64 in Appendix B.2, this problem is NP-complete in general. Therefore, we consider a convex relaxation of it. We use an entropic regularizer $h(M) = \sum_{i,j=1}^{n} M_{ij} \log(M_{ij})$, which makes the objective strongly convex and enforces positivity of M. This regularizer appears in a Bregman divergence term $D_h(M; M_0) = h(M) - h(M_0) - \langle \nabla h(M_0), M - M_0 \rangle$, which can be used to ensure the output does not stray too far from an initial guess M_0. Specifically, we consider the problem

$$\begin{aligned} \min_{M} \quad & \frac{1}{2} \operatorname{trace}(MA) + \mu D_h(M; M_0) \\ \text{subject to} \quad & M_{ij} = m_{ij} \quad \forall\, (i,j) \in \mathcal{K} \\ & n_{\min} \mathbb{1}_n \leq M \mathbb{1}_n \leq n_{\max} \mathbb{1}_n \\ & n_{\min} \mathbb{1}_n \leq M^T \mathbb{1}_n \leq n_{\max} \mathbb{1}_n \ , \end{aligned} \tag{3.6}$$

where the values m_{ij} for $i, j \in \mathcal{K} \coloneqq (\mathcal{S} \times \mathcal{S}) \cup \{(1,1), \ldots, (n,n)\}$ represent the known entries of M. In Appendix B.3 we discuss an alternative relaxation of the labeling problem that was proposed by Bach and Harchaoui (2007).

Optimizing the dual of problem (3.6) via alternating minimization (see Appendix B.2), we obtain Algorithm 4. Note that in the case where the cluster sizes are predetermined and no entries of M are known, this reduces to the Sinkhorn-Knopp algorithm. The matrix balancing algorithm can be analyzed using the proof technique of Soules (1991). We detail in Appendix B.2 the computation of the Jacobian driving the iteration process towards a fixed point. In practice we find that 10 steps of the alternating minimization suffice.

Algorithm 4 Matrix Balancing

Inputs: Matrix $A \in \mathbb{R}^{n \times n}$
Matrix $\hat{M} \in \{0, 1, ?\}^{n \times n}$ encoding known relations $m_{i,j} \in \{0, 1\}$ with $(i, j) \in \mathcal{K}$
Hyperparameters:
Minimum and maximum cluster sizes $n_{\min}$, $n_{\max}$, number of iterations T, entropic regularization μ
Initialize: $\tilde{Q} = \mu^{-1} A - \log(\mathbb{1}_n \mathbb{1}_n^T / k)$
$n_\Delta = (n_{\max} - n_{\min})/2$
$n_\Sigma = (n_{\max} + n_{\min})/2$
$u = v = \mathbb{1}_n$
for $t = 1, \ldots, T$ **do**
$N_{ij} \leftarrow m_{ij}/(u_i v_j)$, $(i, j) \in \mathcal{K}$
$N_{ij} \leftarrow \exp(-\tilde{Q}_{ij})$, $(i, j) \notin \mathcal{K}$
$p_{v,i} \leftarrow \mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma, n_\Delta)} \left(N_{i,\cdot}^T v\right)$, $i = 1, \ldots, n$
$u_i \leftarrow p_{v,i}/(N_{i,\cdot}^T v)$, $i = 1, \ldots, n$
$p_{u,i} \leftarrow \mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma, n_\Delta)} \left(N_{\cdot,i}^T u\right)$, $i = 1, \ldots, n$
$v_i \leftarrow p_{u,i}/(N_{\cdot,i}^T u)$, $i = 1, \ldots, n$
end for
Output: $M = \mathrm{diag}(u) N \,\mathrm{diag}(v)$

Overall algorithm. The overall XSDC algorithm for the case where some labeled data is present is summarized in Algorithm 5. The algorithm proceeds as follows. First, we initialize the parameters V randomly and then optimize the objective on the labeled data to obtain initial estimates of V, W, and b. Next, we proceed to optimize using the labeled and unlabeled data together. At each iteration, we draw a mini-batch of n_b inputs $X^{(t)} = (x_1^{(t)}, \ldots, x_{n_b}^{(t)})$ with corresponding labels $Y^{(t)}$ (some known, some unknown). We compute the output of the network $\Phi_{V^{(t)}}(X^{(t)}) = (\phi_{V^{(t)}}(x_1^{(t)}), \ldots, \phi_{V^{(t)}}(x_{n_b}^{(t)}))^T$ and the corresponding matrix $A_\lambda(\Phi_{V^{(t)}}(X^{(t)})) = \lambda \Pi_{n_b} (\Pi_{n_b} \Phi_{V^{(t)}}(X^{(t)}) \Phi_{V^{(t)}}(X^{(t)})^T \Pi_{n_b} + n_b \lambda \,\mathrm{I})^{-1} \Pi_{n_b}$. We then perform matrix balancing to obtain $M^{(t)}$. Fixing $M^{(t)}$, we then take a gradient step based on the ULR-SGO objective. At the end we obtain labels $\hat{Y}_\mathcal{U}$ for the unlabeled data using 1-nearest neighbor on the feature representations $\Phi_{V^{(T)}}(X)$. Finally, we estimate the parameters W and b by computing the solution to the least squares problem with X and $[Y_\mathcal{S}, \hat{Y}_\mathcal{U}]$. The algorithm for

Algorithm 5 XSDC (when some labeled data is present)

Input: Labeled data $X_{\mathcal{S}}, Y_{\mathcal{S}}$
Unlabeled data $X_{\mathcal{U}}$
Randomly initialized network parameters $V^{(1)}$
Number of iterations T
Initialize:
$V^{(1)}, W^{(1)}, b^{(1)} \leftarrow$ Optimize (3.4) over V, W, b using $X_{\mathcal{S}}, Y_{\mathcal{S}}$, starting from $V^{(1)}$
for $t = 1, \ldots, T$ **do**
    $X^{(t)}, Y^{(t)} \leftarrow$ Draw minibatch of samples
    $M^{(t)} \leftarrow \text{MatrixBalancing}(A_\lambda(\Phi_{V^{(t)}}(X^{(t)})), Y^{(t)}{Y^{(t)}}^T)$
    $V^{(t+1)} \leftarrow \text{ULR-SGO step}(\Phi_{V^{(t)}}(X^{(t)}), M^{(t)}, V^{(t)})$
end for
$\hat{Y}_{\mathcal{U}} \leftarrow \text{NearestNeighbor}(\Phi_{V^{(T)}}(X), Y_{\mathcal{S}})$
$\hat{W}, \hat{b} \leftarrow \text{RegLeastSquares}(X, [Y_{\mathcal{S}}, \hat{Y}_{\mathcal{U}}])$
Output: $\hat{Y}_{\mathcal{U}}, V^{(T)}, \hat{W}, \hat{b}$

the case when no labeled data is present is similar; see Appendix B.4.2.

The XSDC algorithm has two significant benefits in addition to working with any amount of labeled and unlabeled data. First, learning the features does not require knowledge of the number of clusters. Instead, it requires only a bound on the fraction of points per cluster, for use in the matrix balancing. Specifying such a bound is easier than providing the number of clusters. The only time we must use knowledge of the number of clusters is when evaluating the performance of the learned features.

Second, the algorithm is trivially extendable to the case where we have additional must-link or must-not-link information related to the labels. For example, if we know observations i and j must not have the same label, we can encode that constraint in the above problem by adding (i, j) to $\mathcal{K}$ and setting $m_{ij} = 0$. The algorithm itself is otherwise identical. This is an important extension for cases where labelers may not have been able to identify the correct label for an observation (e.g., "Welsh springer spaniel") but could provide certain relevant label information (e.g., the dog is the same breed as the dog in another image).

3.5 Experiments

The framework we proposed may be applied to any amount of labeled and unlabeled data. In the experiments we illustrate how the proposed approach can be used for end-to-end learning when few labels are known. Our goal is not to obtain state-of-the-art results in any specific application domain. Rather, we aim to show that our method is able to successfully leverage unlabeled data when labeled data is scarce. We show that when additional labeled data is unavailable but unlabeled data is plentiful we can typically use the unlabeled data to improve the classification accuracy.

We would expect our approach to improve if domain-specific tricks were used. However, exploring specialized versions of our algorithm for specific applications is beyond the scope of this chapter. Instead, we focus on unifying learning with no labeled data, some labeled data, and fully labeled data in a single training objective. We do this in a domain-agnostic manner.

3.5.1 Choice of ϕ

One benefit of the XSDC algorithm is that it can learn a similarity measure for clustering. Typical clustering methods either do not transform the features or use a kernel-based method. However, clustering in the original feature space is often ineffective. Moreover, clustering using the Gram matrix on the inputs is infeasible when there are a large number of observations and ineffective when the kernel is improperly chosen (Perez-Cruz and Bousquet, 2004). In this work we therefore consider kernel networks for ϕ that are trained to approximate a kernel at each layer.

Many methods for approximating kernels exist, including random Fourier features and the Nyström method (Rahimi and Recht, 2007; Williams and Seeger, 2000; Mohri et al., 2012). Random Fourier features are data-independent and the parameters of the Nyström method are typically selected at random or via quantization (Oglic and Gärtner, 2017). We will instead learn the parameters of the Nyström method, similarly to Mairal (2016). The regularized Nyström method approximates a kernel k by computing the inner products of

features $\phi(x)$ defined by $\phi(x) = (k(V^T V) + \epsilon I)^{-1/2} k(V^T x)$ for some small $\epsilon > 0$ where the matrix V contains the parameters.

We expect similar behavior for other kinds of networks, given observations made by Lee et al. (2018); Matthews et al. (2018), and Belkin et al. (2018), and our observations in Chapter 2.

3.5.2 Experimental details

Experimental setup. The experiments focus on four datasets: the vectorial datasets Gisette (Guyon et al., 2004) and MAGIC (Bock et al., 2004) and the image datasets MNIST (LeCun et al., 2001) and CIFAR-10 (Krizhevsky and Hinton, 2009). The sizes and dimensions of each dataset may be found in Table B.1 in Appendix B.4. For more information regarding the dataset splits and how the data was transformed, see Appendix B.4.

The architectures we use in the experiments are kernel networks. For the vectorial datasets we use single-layer kernel networks (KNs) that approximate a Gaussian RBF kernel using the Nyström method. In contrast, for MNIST we use a convolutional kernel network (CKN) translation of LeNet-5 (LeCun et al., 2001) and for CIFAR-10 we use a CKN applied to the gradient map on the inputs (CKN-GM) (Mairal et al., 2014b). For each of these networks we use 32 filters per layer for the hidden layers. These architectures and datasets were chosen because they represent a broad spectrum in terms of performance. For details on the parameter values and hold-out validation, see Appendix B.4. The hold-out validation is performed on the datasets for each quantity of labeled data but with a single random seed. The best parameters found are used for all other random seeds.

Training. The training is performed as follows. The network parameters are initialized by randomly sampling from the feature representations at each layer of the network. Then the network is trained for 100 iterations using the labeled data. Finally, the network is trained using the ULR-SGO algorithm and matrix balancing on the labeled and unlabeled data for 400 iterations. Unless otherwise specified, $n_{\min} = n_{\max}$ in the matrix balancing, i.e., all classes

are assumed to be equally represented within each mini-batch. We evaluate the performance of the learned representations every 10 iterations.

Code. The code for this project is written using Faiss, PyTorch, SciPy, and the YesWeCKN code from Chapter 2 (Johnson et al., 2019; Paszke et al., 2019; Virtanen et al., 2020). It may be found online at `https://github.com/cjones6/xsdc`.

3.5.3 Results

Improvement with unlabeled data. In the experiments we first compare the XSDC algorithm to two simple baselines: an initial supervised training of the classifier when the network has random weights ("random initialization") and an initial supervised training of both the network and the classifier ("supervised initialization"). In the latter case the network is trained on only the labeled data. In both cases, when evaluating the performance the labels of the unlabeled data are first estimated using 1-nearest neighbor with the labeled data based on the learned features. The classifier is then trained on the labeled and unlabeled data. The reported accuracy of the supervised initialization is the test accuracy after 100 iterations. In contrast, the reported accuracy of XSDC when labeled data exists is the test accuracy observed at the iteration where the validation accuracy is highest. We report this value because the algorithm can overfit before 500 iterations. In the case where no labeled data exists we report the highest observed test accuracy. We performed 10 trials when varying the random seed and report the mean and standard deviation of the corresponding results.

We would expect that XSDC would provide an improvement over the supervised initialization when there are gains to be had from additional labeled data. Otherwise, we would expect training on additional unlabeled data to provide little to no benefit. This is what we see in Figure 3.3. Figure 3.3 compares the accuracy of the XSDC algorithm to the initializations as the quantity of labeled data varies. From all of the plots we can see that the performance of XSDC relative to the supervised baseline is much larger when the quantity of labeled data is smaller. With 50 labeled examples the accuracy on Gisette increases by 5% on average

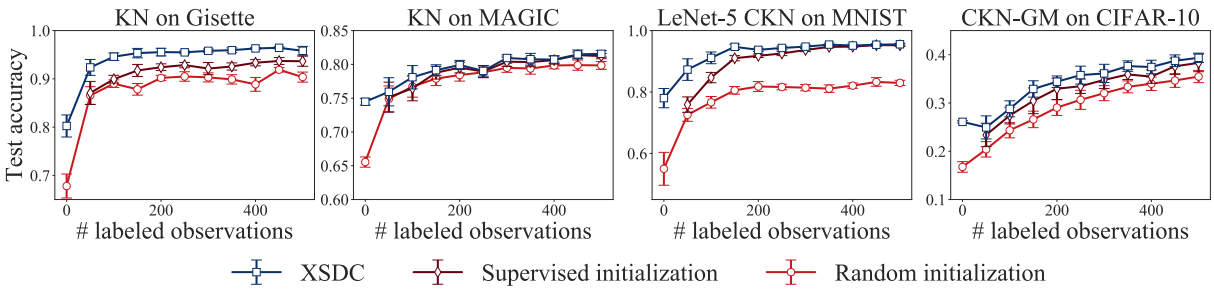


Figure 3.3: Average performance across 10 trials of XSDC when varying the quantity of labeled data. The error bars show one standard deviation from the mean.

when using XSDC instead of the supervised baseline. On MAGIC the gain is more modest, at 1%. For MNIST the gain is 13%, while for CIFAR-10 it is 5%. In contrast, for 500 labeled observations XSDC outperforms the supervised baseline by 2% on Gisette but is only 0.3% better than the baseline on MAGIC. The latter results make sense since the increase in performance of the supervised initialization with the quantity of labeled data has started leveling off by then. On MNIST the improvement when there are 500 labeled observations drops to 0.3%, while on CIFAR-10 it drops to 1.1%. Note that the drop in accuracy of XSDC on CIFAR-10 from zero to 50 labeled observations is likely because we report the highest observed test accuracy for the case of zero labeled observations.

There are two other noteworthy aspects of Figure 3.3. First, it shows that XSDC can improve over the unsupervised initialization even in the case where there is no labeled data. The relative improvement in accuracy over the unsupervised baseline ranges from 14% on MAGIC to 56% on CIFAR-10 when no labeled data is present. Second, the standard deviation of the difference in the performance between the supervised baseline and XSDC tends to be larger when the gap in the performance between XSDC and the supervised baseline is larger, as expected. For example, on Gisette the standard deviation of the difference in the performance of XSDC and the supervised baseline is 1.6% in the case of 50 labeled observations, but only 0.9% in the case of 500 labeled observations.

We also visualize the results, examining the case where 50 images from MNIST are labeled.

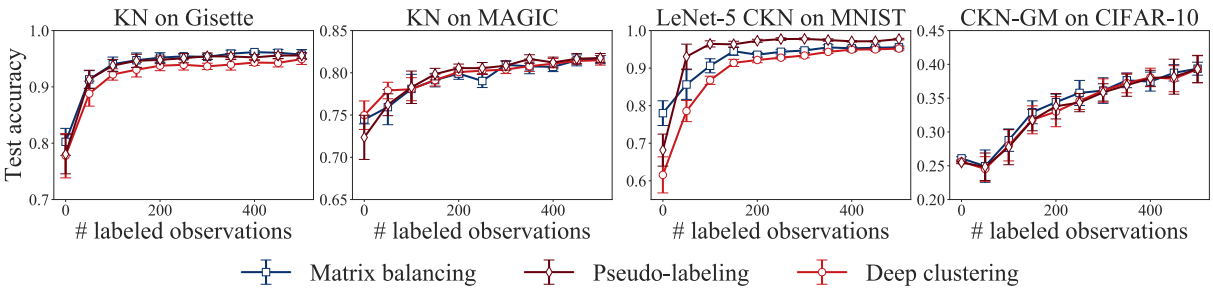


Figure 3.4: Average performance across 10 trials of XSDC with matrix balancing and two alternative labeling methods (pseudo-labeling and deep clustering) when varying the quantity of labeled data. The error bars show one standard deviation from the mean.

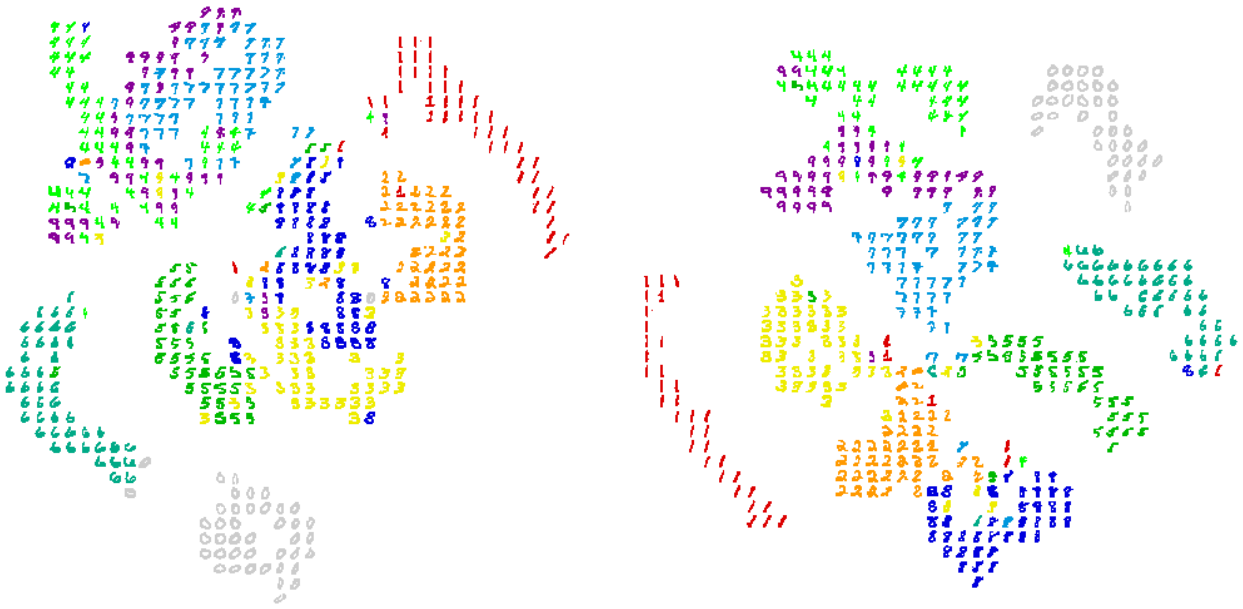
Figure B.2 in Appendix B.5 depicts the feature representations of the unlabeled data at various points of the training process. For each plot the feature representation was projected to 2-D using t-SNE (Van Der Maaten and Hinton, 2008). The observations are color-coded according to their true labels. For more details regarding the visualization, see Appendix B.5. Comparing Figures B.2c and B.2d, we can see that XSDC tends to increase the separation between clusters relative to the supervised initialization. The digits 4, 7, and 9 are a bit less separated. However, the digits 5 and 8 are each generally all in one cluster after running XSDC.

Comparison to alternative labeling methods. Next, we compare to two alternative labeling methods: pseudo-labeling (Lee, 2013) and deep clustering (Caron et al., 2018). Pseudo-labeling is a method designed to learn feature representations from labeled data and unlabeled data. Label assignment is performed by predicting labels from regression on the learned features. In contrast, deep clustering is a method designed to learn feature representations from unlabeled data and assign labels to unlabeled data. Label assignment is performed by k-means clustering with the learned features. Designing a variant working with both labeled data and unlabeled data was beyond the scope of Caron et al. (2018). See Appendix B.4.4 for how we adapted pseudo-labeling and deep clustering to the unsupervised setting and the semi-supervised setting, respectively.

Figure 3.4 displays results comparing the labeling method in XSDC (matrix balancing) to the labeling methods from pseudo-labeling and deep clustering. From the plots we can see that the accuracy with matrix balancing and pseudo-labeling are only significantly different when training the LeNet-5 CKN on MNIST. However, both matrix balancing and pseudo-labeling typically outperform deep clustering when training the kernel network on Gisette and the LeNet-5 CKN on MNIST. On average, matrix balancing is 0.8-3% better than deep clustering when training the kernel network on Gisette and 0.3-21% better than deep clustering when training the LeNet-5 CKN on MNIST. These results suggest that for certain architectures and datasets, using label information may be essential to achieving a performance close to the best possible one. On the other hand, the choice of how that label information is incorporated, whether it is by matrix balancing or pseudo-labeling, may matter less frequently in terms of the performance.

Improvement with additional constraints. As noted in Section 3.4.2, XSDC can seamlessly incorporate additional must-link and must-not-link constraints. To assess the benefit of adding such constraints, we provide additional experiments with the LeNet-5 CKN on MNIST. We consider two forms of additional constraints: (1) Must-not-link constraints derived from knowledge of whether or not each unlabeled observation was from either class 4 or 9; and (2) Random correct must-link and must-not-link constraints among pairs of unlabeled observations and random correct must-not-link constraints between pairs of unlabeled and labeled observations. The pairs of classes in (1) were selected because they are frequently confused. This attempts to mimic a situation in which a labeler knows that an observation belongs to one of two classes, but is not sure which one. Each random constraint in (2) was added with probability 1/3, yielding approximately the same number of constraints as (1). See Appendix B.4.5 for additional details.

Figure 3.5b visualizes the feature representations resulting from constraints of the form (1) for the case of 50 labeled observations from MNIST. Examining this figure, we can see that the clusters are generally well-separated, including the bright green, light blue, and



(a) XSDC

(b) XSDC with additional constraints

Figure 3.5: Visualizations of the unlabeled MNIST features obtained when training the LeNet-5 CKN with 50 labeled observations. The CKN features were projected to 2-D using t-SNE. The constraints used in (b) were derived from knowledge of whether the label for each unlabeled point lay in the set {4,9}.

purple clusters, which correspond to the digits 4, 7, and 9, respectively. Visually, this is an improvement over the t-SNE projections when the additional constraints are not used (*cf.* Figure 3.5a).

Finally, Figure B.3 in Appendix B.5 displays results comparing the test accuracy on MNIST when including and not including the additional constraints on the labels. As expected, adding the additional constraints generally improves the performance. The addition of random correct constraints results in the best performance, likely because these provide more knowledge related to the difficult-to-distinguish classes.

Performance with unbalanced data. The XSDC algorithm can handle unbalanced datasets by changing the bounds on the cluster sizes in the matrix balancing algorithm. To

present an example of how XSDC performs on unbalanced unlabeled data we again trained the LeNet-5 CKN on MNIST. We used 50 labeled observations, equally distributed across classes. For the unlabeled data we varied the fraction of labels 0-4 and the fraction of labels 5-9 between 5% and 95%. For training we use the hold-out validation set to determine the bounds on the cluster sizes.

The results are presented in Figure B.3b in Appendix B.5. Training with XSDC on both the labeled and unlabeled data is nearly always better than training on the labeled data only (dashed curve). As expected, the performance tends to be better for more balanced data. The best accuracy was 85%, obtained with 40% 0-4's, while the worst accuracy was 64%, obtained with 95% 0-4's. In contrast, the accuracy when training on only the labeled data was 69%. These results suggest that as long as one believes that the unlabeled data is not extremely unbalanced, it could be beneficial to use it during training.

Sensitivity to hyperparameters. The XSDC algorithm generally has four or five hyperparameters to tune in the semi-supervised case (depending on the network). In order to assess the importance of these parameters, we perform a sensitivity analysis, again for the LeNet-5 CKN on MNIST with 50 labeled observations. Figure B.4 in Appendix B.5 displays the results when varying one parameter at a time, fixing the others to their values from hold-out validation. From the plots we can see that the parameter that requires the most careful tuning in this setting is the semi-supervised learning rate. The learning rate for the supervised initialization, along with the penalties on the centered features and classifier weights, just need to be sufficiently small.

3.6 Conclusion

In this work we presented a principled learning algorithm called XSDC that can be used on any amount of labeled and unlabeled data. In the special case of unsupervised learning the objective is a clustering objective in which the feature representation is also learned. In contrast, in the special case of supervised learning, the objective is a classification objective.

We demonstrated the effectiveness of XSDC on four datasets, showing that when adding additional labeled data would help, substituting it with unlabeled data still often yields large performance improvements.

Chapter 4

REPRESENTATION LEARNING FOR CHANGE-POINT ESTIMATION

Joint work with Z. Harchaoui.[1]

Abstract. We propose an approach to retrospective change-point estimation that includes learning feature representations from data. The feature representations are specified within a differentiable programming framework, that is, as parameterized mappings amenable to automatic differentiation. The proposed method uses these feature representations in a penalized least-squares objective into which known change-point labels can be incorporated. We propose to minimize the objective using either an alternating optimization procedure or a smoothing procedure. We present numerical illustrations on synthetic and real data showing that learning feature representations can result in more accurate estimation of change-point locations.

4.1 Introduction

The problem of detecting changes in the distribution of a sequence of observations arises in a wide variety of disciplines. Change-point algorithms have, for example, been used to detect seizures based on EEG data (Gardner et al., 2006), to spot attacks on computer networks (Tartakovsky et al., 2006), to estimate the volatility of financial data (Spokoiny, 2009), and to perform video summarization (Potapov et al., 2014). When the input observations are complex structures, such as graphs and images, the success of change-point estimation

[1]This chapter is an expanded version of the work of Jones and Harchaoui (2020). © 2020 IEEE. Reprinted, with permission, from C. Jones and Z. Harchaoui. End-to-End Learning for Retrospective Change-Point Estimation. *IEEE Workshop on Machine Learning for Signal Processing (to appear)*, 2020. We gratefully acknowledge support from NSF DMS 1810975 and faculty research awards.

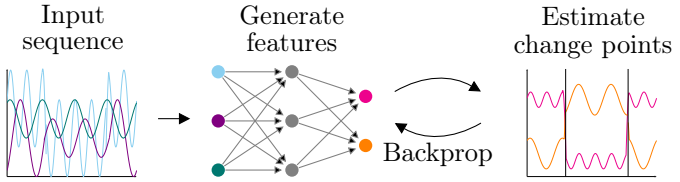


Figure 4.1: Illustration of the algorithm. For a given input sequence, features are generated by separately applying a (deep) network to the vector input at each time point. The change points are then estimated. The gradient of the resultant objective is subsequently computed via backpropagation and the network parameters are updated.

methods relies heavily on the feature representation of the data. Manually designing effective feature representations can be an arduous task. However, most of the current change-point literature assumes that the feature representations are fixed and focuses on various ways of detecting changes.

In this chapter we provide a framework for learning feature representations while performing retrospective change-point estimation. We introduce a penalized least-squares objective allowing us to learn expressive features from training data in order to better perform change-point estimation on new sequences. The resultant approach, depicted in Figure 4.1, allows for the features to be learned regardless of the level of supervision in the context of change-point problems: having no labeled change points is not a problem, but the objective can make use of any number of labeled change points that exist in an effort to improve the feature representations. The optimization can be performed using an alternating approach. However, in Section 4.5 we show how to optimize a smoothed version of the objective. Optimizing the smoothed objective could potentially yield faster convergence in some applications. In Section 4.6 we demonstrate that learning the feature representations can improve the estimated segmentations and we explore the effectiveness of smoothing the objective.

4.2 Related Work

Change-point methods can be applied in two settings: the online setting and the offline setting (Basseville and Nikiforov, 1993; Chen and Gupta, 2012; Kay, 1993). We focus on the offline setting, in which one performs the change-point task "retrospectively", examining all of the data at once and then estimating the change-point locations. Our framework employs ideas from three different research threads: kernel-based retrospective change-point estimation, semi-supervised learning, and smoothing for structured prediction.

4.2.1 Nonparametric change-point estimation

While the retrospective change-point literature is vast, relatively little work has been performed on detecting changes in structured data. In the structured data regime, the framework of choice is generally kernel-based methods. Kernel-based methods for detecting a single change are typically based on two-sample tests, including tests based on the maximum mean discrepancy or variations thereof (Gretton et al., 2012; Zaremba et al., 2013; Li et al., 2019), and tests based on the kernel Fisher discriminant ratio (Harchaoui et al., 2008). On the other hand, kernel-based methods for estimating multiple change points are often based on minimizing the sum of unnormalized variances within segments (Harchaoui and Cappé, 2007). The difficulty with kernel-based methods is that the kernel needs to be chosen appropriately based on the data, and it is often not clear a priori what kernel will work well. We can solve this problem with our framework by learning the parameters of a kernel or kernel network.

The non-smooth, non-convex nature of our objective function makes optimizing it with traditional methods more difficult. We propose to learn the features by smoothing the objective function, similarly to what Pillutla et al. (2019) did for structured support vector machines. We prove bounds on the error from using the smoothed objective rather than the original objective.

4.2.2 Towards representation learning

The idea of learning feature representations for change-point estimation has received little attention. Kim et al. (2009) argued that labeled change points can be used not only for evaluation purposes but also for learning purposes. Soon thereafter, Saatcci et al. (2010) introduced a Bayesian online change-point detection method in which time series are modeled by Gaussian processes. They showed how the parameters of the Gaussian processes could be learned from data. Later, Hoai and De la Torre (2014) and Sangnier et al. (2016) proposed learning classifiers for detecting changes as quickly as possible following an event. In addition, Lajugie et al. (2014) proposed learning a Mahalanobis metric in a large-margin framework for change-point estimation.

Recently a few papers have advanced the idea of training deep networks for change-point estimation. Learning a deep feature representation by maximizing the power of the maximum mean discrepancy test statistic was considered by Chang et al. (2019). However, the objective function resulted from several approximations, so it is unclear whether the method indeed maximizes a bound on the power. On the other hand, Nagano et al. (2019) took a Bayesian approach. The authors proposed learning a hierarchical Dirichlet process–Variational autoencoder–Gaussian process–Hidden semi-Markov model. One main downside to this approach is that the computation time is cubic in the length of a sequence, making it impractical for long sequences. In comparison, the runtime of our approach is quadratic in the length of a sequence.

In contrast to the above papers, we propose a single objective function for learning deep feature representations and retrospectively estimating change points in a frequentist manner regardless of the amount of labeled data.

4.3 Change-Point Estimation with Any Level of Supervision

The goal of retrospective change-point estimation is to locate changes in the distribution of a sequence of observations. Concretely, consider a sequence of independent random

variables $X_t \sim P_t$ for some distributions P_t for $t = 1, \ldots, T$, with corresponding realizations $x_1, \ldots, x_T$. Change-point estimation aims to estimate the locations $t_1, \ldots, t_m$ of m changes in the distributions P_t over $t = 1, \ldots, T$. Throughout this chapter the number of changes m is assumed to be known.

Two general ways of approaching this problem are: (1) propose a model and fit it to the data; and (2) perform constrained clustering. In this section we first review these two approaches. We then discuss how the latter approach can be extended in the case where some of the change points in a sequence are known.

4.3.1 Change-point estimation via model fitting

Model-based change-point estimation algorithms generally model the data within each segment $j = 0, \ldots, m$ via a function f_j potentially depending on $X_1, \ldots, X_T$, t, and parameters θ, in addition to a mean-zero error term ϵ_t. In other words,

$$X_t = f_j(X_1, \ldots, X_T, t; \theta) + \epsilon_t \quad \forall \quad t = t_j, \ldots, t_{j+1} - 1 \ ,$$

where we define $t_0 = 1$ and $t_{m+1} = T + 1$. Typically the ϵ_t's are independent and identically distributed.

Table 4.1 provides some examples of frequentist change-point models. Bayesian models have also been proposed, including the piecewise polynomial regression model of Fearnhead and Liu (2011) and the Dirichlet process hidden Markov model of Ko et al. (2015). Next we review the derivations of the objectives for two example models: the normal change-in-mean model and the change-in-slope model.

Example 5 (Normal change-in-mean model)**.** *For $x \in \mathbb{R}^d$ let $f_j(X_1, \ldots, X_T, t; \theta) = \mu_j$ and $\epsilon_t \sim N(0, \sigma^2 \, \mathrm{I}_d)$. Then the log-likelihood (ignoring the constant term) is given by*

$$\ell(t_1, \ldots, t_m, \mu_0, \ldots, \mu_m, \sigma^2; x_1, \ldots, x_T) = -\frac{1}{2\sigma^2} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \|x_t - \mu_j\|_2^2 - dT \log(\sigma) \ .$$

The maximum likelihood estimates of the change points are independent of σ^2*. Moreover, optimizing over* μ_j *for all* j *yields the empirical means* $\hat{\mu}_j = 1/(t_{j+1} - t_j) \sum_{t=t_j}^{t_{j+1}-1} x_j$*. Hence, the change-point estimation problem can be written as*

$$\min_{t_1,\ldots,t_m} \quad \frac{1}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \|x_t - \hat{\mu}_j\|_2^2 \,. \tag{4.1}$$

This problem can be solved efficiently using dynamic programming (Fisher, 1958; Bellman, 1961; Kay, 1993).

Example 6 (Change-in-slope model)**.** *For* $x \in \mathbb{R}$ *let* $f_j(X_1, \ldots, X_T, t; \theta) = a_j + \frac{a_{j+1}-a_j}{t_{j+1}-t_j}(t - t_j)$*, where the* a_j*'s for* $j = 0, \ldots, m+1$ *are parameters to be learned. Furthermore, assume that the* ϵ_t*'s have mean zero and variance* σ^2*. Here* a_j *is the fitted value at the beginning of segment* j *for all* j *and* $(a_{j+1} - a_j)/(t_{j+1} - t_j)$ *is the estimated slope of segment* j*. Note that this model constrains the estimated linear function for segment* j *to coincide with the estimated linear function for segment* $j + 1$*.*

Using the square loss leads to the problem

$$\min_{t_1,\ldots,t_m,a_0,\ldots,a_{m+1}} \quad \frac{1}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \left(x_t - a_j - \frac{a_{j+1} - a_j}{t_{j+1} - t_j}(t - t_j) \right)^2 \,.$$

This problem can also be solved efficiently using dynamic programming (Fearnhead et al., 2019).

4.3.2 Change-point estimation via constrained clustering

Alternatively, we can view change-point estimation as a constrained clustering problem. To see this, note that the change points induce a segmentation of the observations, with segments $x_{t_j}, \ldots, x_{t_{j+1}-1}$ for $j = 0, \ldots, m$. Each such segment can be viewed as a cluster. As each segment must consist of contiguous observations, this contiguity constraint needs to be incorporated when clustering.

Model	$f_j(X_1, \ldots, X_T, t; \theta)$	Distribution of ϵ_t	Estimation method
Normal change in mean	μ_j	$N(0, \sigma^2 \mathrm{I}_d)$	Max. likelihood
Kernel change in mean	μ_j	Centered: $\mathbb{E}[\langle \epsilon_i, g \rangle_{\mathcal{H}}] = 0 \; \forall i$	Least squares
Normal change in mean or variance	μ_j	$N(0, \Sigma_j)$	Max. likelihood
Change in slope	$a_j + \frac{a_{j+1} - a_j}{t_{j+1} - t_j}(t - t_j)$	Mean 0, variance σ^2	Least squares
VAR(p) change in parameters	$\mu_j + \sum_{r=1}^{p} A_{jr} X_{t-r}$	Mean 0, covariance Σ_j	Quasi-max. likelihood

Table 4.1: Examples of models used in change-point estimation (Quandt, 1958; Arlot et al., 2019; Fearnhead et al., 2019; Bai, 2000). Here it is assumed that X_t belongs to segment j for some j. In each model the errors ϵ_t are assumed to be independent. The parameters $a_j \in \mathbb{R}$ for all j and $A_{j,p} \in \mathbb{R}^{d \times d}$ for all j, p.

From this viewpoint, we can consider modifying existing clustering algorithms to produce segmentation algorithms. Define the matrix of observations $x = [x_1, \ldots, x_T]^T$. Let $y = [y_1, \ldots, y_T]^T$ with $y_t \in \mathbb{R}^{m+1}$ be a matrix consisting of one-hot vectors indicating the segment number j to which each observation is assigned. In addition, denote by A^+ the Moore-Penrose pseudo-inverse of a matrix A. In Chapter 3 we defined the following family of clustering problems:

Definition 9 (M-family of clustering problems)**.** *Let $\Psi_\theta : \mathbb{R}^{T \times d} \to \mathbb{R}^{T \times T}$ and $\Gamma_\theta : \mathbb{R}^{T \times d} \to \mathbb{R}^{T \times T}$ be functions parameterized by $\theta \in \mathbb{R}^{d_\theta}$ for some d_θ. The M-family of clustering problems is then given by*

$$
\begin{aligned}
\min_{y} \quad & \operatorname{trace}\left[\Psi_\theta(x)^T (\Gamma_\theta(x) y y^T \Gamma_\theta(x))^\alpha (\Gamma_\theta(x) y y^T \Gamma_\theta(x))^{+\beta}\right] \\
\textit{subject to} \quad & y \mathbb{1}_{m+1} = \mathbb{1}_T \\
& \lambda_1 \left(y^T \mathbb{1}_T - n_{\min} \mathbb{1}_{m+1}\right) \geq 0 \\
& \lambda_2 \left(y^T \mathbb{1}_T - n_{\max} \mathbb{1}_{m+1}\right) \leq 0 \\
& y_{tj} \in \{0, 1\} \quad \forall\, t, j \;,
\end{aligned}
$$

where $\alpha, \beta, \lambda_1, \lambda_2 \in \{0, 1\}$ and $n_{\min}, n_{\max} > 0$.

This family includes the objectives of methods such as k-means and several variants of spectral clustering. We can obtain an analogous family of segmentation problems by further

restricting the matrix y to enforce the cluster contiguity constraint:

Definition 10 (M-family of segmentation problems)**.** *Let* $\Psi_\theta : \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times T}$ *and* $\Gamma_\theta : \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times T}$ *be functions parameterized by* $\theta \in \mathbb{R}^{d_\theta}$ *for some* d_θ*. The* M*-family of segmentation problems is then given by*

$$\begin{aligned} \min_{y} \quad & \operatorname{trace}\left[\Psi_\theta(x)^T(\Gamma_\theta(x)yy^T\Gamma_\theta(x))^\alpha(\Gamma_\theta(x)yy^T\Gamma_\theta(x))^{+\beta}\right] && (4.2)\\ \textit{subject to} \quad & y\mathbb{1}_{m+1} = \mathbb{1}_T \\ & \lambda_1\left(y^T\mathbb{1}_T - n_{\min}\mathbb{1}_{m+1}\right) \geq 0 \\ & \lambda_2\left(y^T\mathbb{1}_T - n_{\max}\mathbb{1}_{m+1}\right) \leq 0 \\ & 0 \leq v^T(y_{t+1} - y_t) \leq 1 \quad \forall\, t = 1,\dots,T-1 && (4.3)\\ & y_{tj} \in \{0,1\} \quad \forall\, t, j\ , \end{aligned}$$

where $\alpha, \beta, \lambda_1, \lambda_2 \in \{0,1\}$*,* $n_{\min}, n_{\max} > 0$*, and* $v = [1, 2, \dots, m+1]^T$*.*

The constraint (4.3) ensures that the segment index j of adjacent observations x_t, x_{t+1} must either be the same or increase by one from t to $t+1$. Next we provide two examples of change-point estimation problems in this family.

Example 7 (Constrained k-means)**.** *Define* $\Psi_\theta(x) = -xx^T$ *and* $\Gamma_\theta(x) = \mathrm{I}_T$*, and let* $\alpha = \beta = 1$ *and* $\lambda_1 = \lambda_2 = 0$*. The resultant* M*-family problem is given by*

$$\begin{aligned} \min_{y} \quad & \operatorname{trace}\left[-xx^T(yy^T)(yy^T)^+\right] \\ \textit{subject to} \quad & y\mathbb{1}_{m+1} = \mathbb{1}_T \\ & 0 \leq v^T(y_{t+1} - y_t) \leq 1 \quad \forall\, t = 1,\dots,T-1 \\ & y_{tj} \in \{0,1\} \quad \forall\, t, j\ . \end{aligned}$$

Following Example 2 in Chapter 3, the minimizer of this problem is the same as that of the

problem

$$\begin{aligned} \min_{y,\mu} \quad & \|x - y\mu\|_F^2 \\ \textit{subject to} \quad & y\mathbb{1}_{m+1} = \mathbb{1}_T \\ & 0 \leq v^T(y_{t+1} - y_t) \leq 1 \quad \forall\, t = 1, \ldots, T-1 \\ & y_{tj} \in \{0, 1\} \quad \forall\, t, j\ . \end{aligned}$$

From the constraints on y we can rewrite this as $\min_{t_1,\ldots,t_m,\mu_0,\ldots,\mu_m} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \|x_t - \mu_j\|_2^2$. After minimizing over $\mu_0, \ldots, \mu_m$, we can note that the argmin of this objective is the same as that of the following objective:

$$\min_{t_1,\ldots,t_m} \quad \frac{1}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \|x_t - \hat{\mu}_j\|_2^2\ ,$$

where $\hat{\mu}_j = 1/(t_{j+1} - t_j) \sum_{t=t_j}^{t_{j+1}-1} x_j$. This is precisely the objective obtained with the Gaussian change-in-mean model from Section 4.3.1.

Example 8 (Constrained normalized-cut spectral clustering)**.** *Let $S \in \mathbb{R}^{T \times T}$ with $S \geq 0$ be a symmetric similarity matrix derived from x. Define $D = \operatorname{diag}([D_t]_{t=1}^T)$ with $D_t = \sum_{j=1}^{T} S_{tj}$ for all t and $L = D - S$ to be the degree and Laplacian matrices, respectively, associated with x. Define $\Psi_\theta(x) = D^{-1/2} L D^{-1/2}$ and $\Gamma_\theta(x) = D^{1/2}$, and let $\alpha = \beta = 1$ and $\lambda_1 = \lambda_2 = 0$. These choices lead to the M-family problem given by*

$$\begin{aligned} \min_{y} \quad & \operatorname{trace}\left[D^{-1/2} L D^{-1/2} (D^{1/2} y y^T D^{1/2})(D^{1/2} y y^T D^{1/2})^{+}\right] \\ \textit{subject to} \quad & y\mathbb{1}_{m+1} = \mathbb{1}_T \\ & 0 \leq v^T(y_{t+1} - y_t) \leq 1 \quad \forall\, t = 1, \ldots, T-1 \\ & y_{tj} \in \{0, 1\} \quad \forall\, t, j\ . \end{aligned}$$

Following Example 3 in Chapter 3, we can rewrite this problem as

$$\begin{aligned} \min_{\mathscr{C}=\{C_0,\ldots,C_m\}} \quad & \sum_{j=0}^{m} \sum_{j' \neq j} \frac{Cut(C_j, C_{j'})}{Vol(C_j)} \\ \textit{subject to} \quad & C_j \cap C_{j'} = \emptyset \quad \forall\, j \neq j' \\ & \cup_{j=0}^{m} C_j = \{1, \ldots, T\} \\ & x_t \in C_j \implies x_{t+1} \in C_j \cup C_{j+1} \quad \forall\, t = 1, \ldots, T-1\ , \end{aligned}$$

where $Cut(C, C') \coloneqq \sum_{t \in C} \sum_{t' \in C'} S_{tt'}$. *This is precisely the multi-way normalized cut clustering problem with additional constraints on the cluster assignments.*

In general, the problems based on this approach can be intractable due to the cluster contiguity constraint $0 \leq v^T(y_{t+1} - y_t) \leq 1 \quad \forall\, t = 1, \ldots, T-1$. This constraint requires the binary equivalence matrix M to be block diagonal. As we will see in Section 4.5, one case where this problem is tractable is the constrained k-means problem. An alternative is to consider a relaxation of the problem. For example, one could relax the integral constraints on the y_t's and penalize $\sum_{t=1}^{T-1} |v^T(y_{t+1} - y_t)|$. One could also use a version of convex clustering with a total variation penalty (Pelckmans et al., 2005; Harchaoui and Lévy-Leduc, 2010).

4.3.3 Incorporating known change points

In this work we are interested in performing change-point estimation in the case where some change points may be known. Knowledge of one or more change points in a sequence provides additional information about the entries of the label matrix y and equivalence matrix $M = yy^T$. Specifically, if we know that there is a change point at a time t, then all observations before time t must lie in a different segment than the observations at and after time t. In other words, in terms of the equivalence matrix, all entries M_{ij} and M_{ji} with $i < t$ and $j \geq t$ must be zero. Note that these are must-not-link constraints, as opposed to must-link constraints that usually come from labeled data in clustering.

By taking into account these additional constraints on y or M we can address three

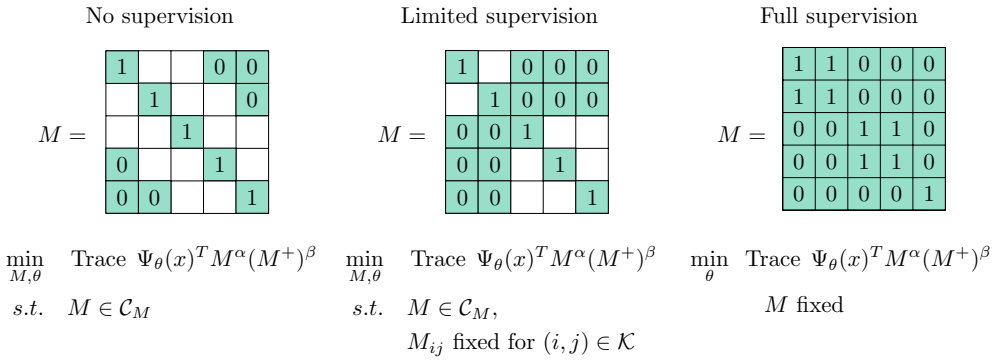


Figure 4.2: Example equivalence matrix M and objective function for varying levels of supervision when the number of change points is known to be two. In the limited supervision case one of the change points, at $t = 3$, is known. For simplicity we set $\Gamma_\theta(x) = \mathrm{I}_T$ in the objective functions. The set $\mathcal{K}$ consists of the indices of the known entries of M.

settings: the unsupervised setting, in which we use change-point estimation to learn y and M in addition to the parameters θ; the semi-supervised setting, in which a combination of change-point estimation and supervised training is used to learn the remaining unknown entries of y and M, in addition to the parameters θ; and the supervised setting, in which supervised training is used to learn the parameters θ.

Figure 4.2 depicts these three regimes for one example in the context of an objective from the M-family of segmentation methods. In this example the number of change points is known to be two and $\mathcal{C}_M = \{M \in \{0,1\}^{T\times T} : \exists y \in \mathcal{C}_y \text{ s.t. } M = yy^T\}$ with $\mathcal{C}_y = \{y \in \{0,1\}^{T\times m} : y\mathbb{1}_{m+1} = \mathbb{1}_T, \lambda_1\left(y^T\mathbb{1}_T - n_{\min}\mathbb{1}_{m+1}\right) \geq 0, \lambda_2\left(y^T\mathbb{1}_T - n_{\max}\mathbb{1}_{m+1}\right) \leq 0, 0 \leq v^T(y_{t+1} - y_t) \leq 1 \quad \forall\, t = 1,\ldots,T-1\}$. In the case of no supervision, we do not know where these change points lie. However, due to the number of change points we know that the first observation cannot lie in the same segment as either of the last two observations. Similarly, the second observation cannot lie in the same segment as the last observation. In the case of limited supervision in which we know that one change point is at $t = 3$ we add additional constraints that the observations before $t = 3$ cannot be in the same segment as the observations at $t \geq 3$. Finally, in the case of full supervision we know the segmentation of the observations.

4.4 End-to-End Learning Objective

In this section we extend the normal change-in-mean objective to an end-to-end learning objective. The normal change-in-mean objective from Section 4.3 assumes a fixed feature representation. However, as noted in the introduction, we often do not know the best way of representing the data a priori. Moreover, kernel-based methods are often sensitive to the feature representation. With the proposed end-to-end learning objective we will instead learn a feature representation. We call our resultant algorithm XSCPE for X-supervised change-point estimation, where X can be "un", "semi", or "-", similarly to the algorithm from Chapter 3.

4.4.1 Notation

Let $X_t^{(i)} \sim P_{\theta^{t^{(i)}}}$, $t = 1, \ldots, T^{(i)}, i = 1, \ldots, n$ be n sequences of independent random variables from some distributions $P_{\theta^{t^{(i)}}}$ parameterized by $\theta^{t^{(i)}}$. Furthermore, let $x_1^{(i)}, \ldots, x_{T^{(i)}}^{(i)}$ be realizations of $X_1^{(i)}, \ldots, X_{T^{(i)}}^{(i)}$ for all i. For clarity of exposition we will assume $x_t^{(i)} \in \mathbb{R}^{d_x}$ for all t and some $d_x \in \mathbb{N}_+$. We are interested in (1) obtaining a feature representation for $x_t^{(i)}$ for all t, i in order to (2) estimate the locations of changes in the parameters $\theta^{t^{(i)}}$ over $t = 1, \ldots, T^{(i)}$ for all sequences i. When discussing individual observations or a single sequence we will sometimes drop the index i denoting the sequence number for readability.

To address point (2) we aim to locate changes in distribution using a penalized least-squares objective. The objective will act on sequences of observations transformed via a learned feature mapping. Learning feature representations from data can result in more powerful feature representations than data-independent ones such as kernel feature maps. While large classes of universal kernels yield injective mean element maps (Christmann and Steinwart, 2010), the corresponding feature representations may still be poor in face of challenging change-point problems. We consider feature representations implemented by a parameterized feature mapping in a differentiable programming framework, allowing us to approach representation learning using first-order optimization algorithms.

Denote by $\phi : \mathbb{R}^{d_x} \times \mathbb{R}^{d_w} \to \mathbb{R}^{d_\phi}$ such a network, which takes as input an observation $x_t \in \mathbb{R}^{d_x}$ and parameters $w \in \mathbb{R}^{d_w}$ for some $d_w \in \mathbb{N}_+$ and outputs a feature vector $\phi(x_t; w) \in \mathbb{R}^{d_\phi}$ for some $d_\phi \in \mathbb{N}_+$. We will simultaneously learn w and locate changes in the mean of the distributions from which $\phi(x_1^{(i)}; w), \ldots, \phi(x_T^{(i)}; w)$ are drawn. We will assume that there are $m^{(i)}$ such changes, segmenting sequence i into $m^{(i)} + 1$ parts. We denote the change-point locations by $t_1^{(i)}, \ldots, t_{m^{(i)}}^{(i)}$ and define $t_0^{(i)} \equiv 1$ and $t_{m^{(i)}+1}^{(i)} \equiv T^{(i)} + 1$.

4.4.2 Problem formulation

Consider the change-in-mean problem (4.1) where instead of the x_t's we use the feature representations $\phi(x_t; w)$. For fixed w this is then a quasi-likelihood objective derived from the supposition that $\phi(X_t^{(i)}; w) \sim N(\mu^{t^{(i)}}, \sigma^2 \, \mathrm{I}_{d_\phi})$ for some $\mu^{t^{(i)}} \in \mathbb{R}^{d_\phi}$ for all t, i and some $\sigma^2 \in \mathbb{R}_+$ (White, 1982). Simply maximizing the average of these quasi-likelihood objectives over all sequences i with respect to w typically results in the observations $x_t^{(i)}$ all being mapped to the same value, i.e., $\phi(x_t^{(i)}; w) = c$ for some $c \in \mathbb{R}^{d_\phi}$ and for all t and i.

To overcome this problem, we consider constraining the covariance of each sequence. Note that if, for a given sequence, $\phi(x_t; w) = c$ for some c and for all t, then the covariance of the feature representations $\phi(x_t; w)$, $t = 1, \ldots, T$ is zero. When performing change-point estimation we aim to minimize the sum of the *intra*-segment variances for each feature. However, if change points exist then we would expect the sum of the *inter*-segment variances for each feature to be non-zero.

In order to enforce non-zero inter-segment variances we will require that the trace of the global empirical covariance matrix for each sequence be sufficiently large. Observe that for sequence i the trace of the global empirical covariance matrix can be written as

$$\operatorname{trace}\left(\frac{1}{T^{(i)}}\sum_{t=1}^{T^{(i)}}(\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)})(\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)})^T\right) = \frac{1}{T^{(i)}}\sum_{t=1}^{T^{(i)}} \|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\|_2^2 \,,$$

where $\hat{\mu}^{(i)} = 1/T^{(i)} \sum_{t=1}^{T^{(i)}} \phi(x_t^{(i)}; w)$. Now define $\mathcal{T}^{(i)} = \{t_1^{(i)}, \ldots, t_{m^{(i)}}^{(i)}\}$ to be the set of change

points for sequence i. Moreover define the set of all possible sets of change points for sequence i as $\mathcal{C}^{(i)}(m^{(i)}) \coloneqq \{\mathcal{T} = \{t_1, \ldots, t_{m^{(i)}}\} : t_{j+1} - t_j \geq \Delta \; \forall \, j = 0, \ldots, m^{(i)}, \mathcal{T} \supseteq \mathcal{K}^{(i)}\}$ where Δ is the minimum acceptable segment length and $\mathcal{K}^{(i)}$ is the (possibly empty) set of known change points for sequence i. Consider the problem

$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{T^{(i)}} \min_{\mathcal{T} \in \mathcal{C}^{(i)}(m^{(i)})} \sum_{j=0}^{m^{(i)}} \sum_{t=t_j^{(i)}}^{t_{j+1}^{(i)}-1} \left\| \phi(x_t^{(i)}; w) - \hat{\mu}_j^{(i)} \right\|_2^2$$

$$\text{subject to} \quad \sqrt{\frac{1}{T^{(i)}} \sum_{t=1}^{T^{(i)}} \|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\|_2^2 + \epsilon} \geq c \quad \forall \, i = 1, \ldots, n \; . \tag{4.4}$$

The constraint (4.4) could alternatively be written in other forms. For example, we could instead use $1/T^{(i)} \sum_{t=1}^{T^{(i)}} \|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\|_2^2 \geq c^2 - \epsilon$. We choose to use the square root version because of the effect it has when the problem is converted from a constrained problem to a regularized problem. In addition, adding ϵ ensures that the gradient of the regularized problem is Lipschitz.

Define the portion of the regularized objective from sequence i with change points $\mathcal{T}^{(i)}$ as

$$\mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) \coloneqq \frac{1}{T^{(i)}} \sum_{j=0}^{m^{(i)}} \sum_{t=t_j^{(i)}}^{t_{j+1}^{(i)}-1} \left\| \phi(x_t^{(i)}; w) - \hat{\mu}_j^{(i)} \right\|_2^2 - \lambda \sqrt{\frac{1}{T^{(i)}} \sum_{t=1}^{T^{(i)}} \|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\|_2^2 + \epsilon} \; , \tag{4.5}$$

where $\lambda \geq 0$. For simplicity we use the same value of λ for each constraint in the formulation above. Note that if the first term is non-zero and we scale $\phi(x_t^{(i)}; w)$ by a constant a, then as $|a| \to \infty$, $\mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) \to \infty$. On the other hand, if $a \to 0$ then the second term will dominate. This would not be true if we did not square root the trace. However, other transformations, such as the log of the trace, would have a similar effect.

Then, adding an additional penalty on the squared norm of the parameters to promote

smoothness of the learned network ϕ, the regularized problem is given by

$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} \min_{\mathcal{T}^{(i)} \in \mathcal{C}^{(i)}(m^{(i)})} \mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) + \gamma \|w\|_2^2 \,, \tag{4.6}$$

where $\gamma \geq 0$. We will denote the objective in (4.6) by $\mathscr{L}(\{\mathcal{T}^{(i)}\}_{i=1}^n, w; \{x^{(i)}\}_{i=1}^n)$. The next proposition provides conditions under which this problem is bounded below. The proof may be found in Appendix C.1.

Proposition 11. *Assume $\phi(\cdot; w)$ is Lipschitz with constant $L_{\phi_x}(w)$ such that there exist $c_1, c_2 \in \mathbb{R}$ with $0 \leq L_{\phi_x}(w) \leq c_1 + c_2\|w\|_2^2$ for all w. Furthermore, assume the inputs $x_t \in \mathbb{R}^{d_x}$ lie in a bounded set with diameter D according to the Euclidean distance. Then for $\gamma \geq c_2 D\lambda$ the objective* (4.6) *is bounded below by $-\lambda(c_1 D + \sqrt{\epsilon})$.*

The condition $0 \leq L_{\phi_x}(w) \leq c_1 + c_2\|w\|_2^2$ in Proposition 11 applies to all Lipschitz functions $\phi(\cdot; w)$ whose gradients have bounded norm (e.g., convolutional kernel networks whose filters lie on a sphere). It also applies to certain functions, such as two-layer ConvNets with the softplus nonlinearity, whose gradients do not grow too quickly in norm as the norm of w increases.

4.5 Optimization

Problem (4.6) is combinatorial (due to the discrete nature of the change points) and usually non-convex in w (due to the choice of ϕ). In this section we consider two ways of optimizing the parameter w: (a) by alternating between optimizing over the change points $\mathcal{T}^{(i)}$ for all i and optimizing over w; and (b) by optimizing a smoothed version of the objective.

4.5.1 Optimization via alternation

The most straightforward approach to optimizing $\mathscr{L}$ from problem (4.6) is to alternate between optimizing over the change points $\mathcal{T}^{(i)}$ in each sequence i for fixed w using dynamic programming and optimizing over w for fixed $\mathcal{T}^{(i)}$'s using, e.g., gradient descent.

Optimization over the change points. For fixed w, problem (4.6) decomposes into a change-point problem of the form (4.1) for each sequence. When no change points in a sequence are known, optimizing over the change points is a classical problem that can be solved efficiently using dynamic programming. Note that we can write problem (4.1) as

$$\frac{1}{T}\min_{t_m}\left\{s(t_m,t_{m+1})+\min_{t_{m-1}}\left\{s(t_{m-1},t_m)+\cdots+\min_{t_1}\left\{s(t_1,t_2)+s(t_0,t_1)\right\}\cdots\right\}\right\}, \qquad (4.7)$$

where $s(t_j,t_{j+1}) \coloneqq \sum_{t=t_j}^{t_{t+1}-1}\|\phi(x_t;w)-\hat{\mu}_j\|_2^2$ with $\hat{\mu}_j \coloneqq \frac{1}{t_{j+1}-t_j}\sum_{t=t_j}^{t_{j+1}-1}\phi(x_t;w)$. From this we can see that to compute the best segmentation, we can compute the best segmentation when minimizing over t_1 for all possible values of t_2, and then when minimizing over t_2 for all possible values of t_3, etc., up through t_m.

The above observation leads to Algorithm 6 in the classical setting, i.e., where the input $K = 1$ and no change points are known. In this setting Algorithm 6 proceeds as follows. In the first for loop, the partial sums $s(t_0,t_1)$ are computed for all possible values of t_1. Next, for every number of possible change points j, and every possible location of the next change point, t_{j+1}, the change points leading to the smallest partial objective $\min_{t_j}\{s(t_j,t_{j+1})+\cdots+\min_{t_1}\{s(t_1,t_2)+s(t_0,t_1)\}\cdots\}$ are computed. In each case the optimal value of t_j is stored in the matrix I and the corresponding partial objective is stored in the matrix L. Once all of the partial objectives have been computed, the algorithm performs a backward pass, determining from I which indices led to the smallest overall objective value.

In some instances we may know some change points within a sequence but not all of them and want to estimate the remaining unknown change points. Algorithm 6 is able to handle this case via its extension of the classical algorithm. Specifically, when some change points are known, it checks whether t_{j+1} is permissible given the set of known change points $t^1,\ldots,t^{m'}$ for some m'. This entails ensuring (1) having the next change point be t_{j+1} would not skip one of the known change points $t^1,\ldots,t^{m'}$; and (2) having the next change point be t_{j+1} would not lead to the final change point being before the last known change point, $t^{m'}$. The algorithm only allows change points t_{j+1} satisfying these two criteria. This version of

Algorithm 6 with $K = 1$ retains the $O(mT^2)$ complexity of the classical algorithm. The left side of Figure 4.3 shows an example in the case where one change point is known. The case where $K > 1$ is discussed in Section 4.5.2.

Algorithm 6 can be further modified if only ranges $(\ell^0, u^0) \coloneqq (1, 1), (\ell^1, u^1), \ldots, (\ell^{m'}, u^{m'})$, $(\ell^{m'+1}, u^{m'+1}) \coloneqq (T+1, T+1)$ for the locations of known change points are given rather than the exact values. In this case $\min\{t^1, T+1-m\Delta\}$ would become $\min\{\min_{r>0} u^r, T+1-m\Delta\}$ and, if the intervals are non-overlapping, the t's used in computing b_ℓ and b_u would be replaced by ℓ's and u's, respectively.

Update for the network parameters. After optimizing over the change points $\mathcal{T}^{(i)}$ for each sequence i we propose to optimize w. The gradient of the objective $\mathscr{L}$ from problem (4.6) with respect to w is given in the following proposition.

Proposition 12. *Assume that $\phi(x_t^{(i)}; \cdot)$ is differentiable for all t, i. Then the gradient of the objective* (4.6)*, $\mathscr{L}(\{\mathcal{T}^{(i)}\}_{i=1}^n, w; \{x^{(i)}\}_{i=1}^n)$, with respect to w is given by*

$$\nabla_w \mathscr{L}(\{\mathcal{T}^{(i)}\}_{i=1}^n, w; \{x^{(i)}\}_{i=1}^n) = \frac{2}{n} \sum_{i=1}^{n} \frac{1}{T^{(i)}} \sum_{j=0}^{m^{(i)}} \sum_{t=t_j^{(i)}}^{t_{j+1}^{(i)}-1} \left(\nabla_w \phi(x_t^{(i)}; w) \right)^T \tag{4.8}$$

$$\times \left\{ \phi(x_t^{(i)}; w) - \hat{\mu}_j^{(i)} - \frac{\lambda}{2} \left[\frac{1}{T^{(i)}} \sum_{t=1}^{T^{(i)}} \|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\|_2^2 + \epsilon \right]^{-1/2} \left(\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)} \right) \right\} + 2\gamma w \, .$$

If ϕ is sufficiently well-behaved then the gradient of $\mathscr{L}$ is Lipschitz with respect to w. Proposition 13 characterizes conditions under which this is true. The proofs of Propositions 12 and 13 are given in Appendix C.1.

Proposition 13. *Define $\phi_t^{(i)}(w) = \phi(x_t^{(i)}; w)$ for all t, i. Assume $\phi_t^{(i)}(\cdot)$ is differentiable and there exist constants $M_\phi, M_{\nabla_w \phi} \geq 0$ such that for all $x_t^{(i)}$ and w we have $\|\phi_t^{(i)}(w)\|_2 \leq M_\phi$ and $\|\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)}(w))\|_F \leq M_{\nabla_w \phi}$. Moreover, assume there exists a constant $L_{\nabla_w \phi} \geq 0$ such that for all t and i, $\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)})$ is Lipschitz with constant $L_{\nabla_w \phi}$. Then the gradient*

Algorithm 6 Top-K Multiple Change-point Estimation for a Partially-Labeled Sequence

Input: Features $\phi(x_1; w), \ldots, \phi(x_T; w) \in \mathbb{R}^{d_\phi}$
  Number of change points m
  Minimum distance Δ between change points
  Number of segmentations K to be returned (top K)
  Known change points $t^0 \coloneqq 1 < t^1 < \cdots < t^{m'} < t^{m'+1} \coloneqq T + 1$
Perform the forward pass:
for $t_1 = \Delta + 1, \Delta + 2, \ldots, \min\{t^1, T + 1 - m\Delta\}$ **do**
  $L_{1,t_1,1} \leftarrow s(1, t_1)$
  $L_{1,t_1,2:K} \leftarrow \infty$
  $I_{1,t_1,1} \leftarrow 1$
  $I_{1,t_1,2:K} \leftarrow \text{NaN}$
end for
for $j = 1, 2, \ldots, m$ **do**
  for $t_{j+1} = (j + 1)\Delta + 1, \ldots, T + 1 - (m - j)\Delta$ **do**
    heap = MaxHeapify($(\infty, \text{NaN}, \text{NaN}), \ldots, (\infty, \text{NaN}, \text{NaN})$)
    for $t_j = j\Delta + 1, \ldots, t_{j+1} - \Delta$ **do**
      $b_\ell \leftarrow t^{\max\{m'+1-(m-j),0\}}$
      $b_u \leftarrow \min t^r \in \{t^1, t^2, \ldots, t^{m'+1}\}$ s.t. $t^r > t_j$
      if $b_\ell \le t_{j+1} \le b_u$ **then**
        for $k = 1, \ldots, K$ **do**
          HeapPushPop(heap, $(L_{j,t_j,k} + s(t_j, t_{j+1}), t_j, k)$
        end for
      end if
    end for
    top_k = Sort(heap)
    $L_{j+1,t_{j+1},1:K} \leftarrow$ top_k[:][1]
    $I_{j+1,t_{j+1},1:K} \leftarrow$ top_k[:][2] $\times K +$ top_k[:][3] $- 1$
  end for
end for
Perform the backward pass:
$t_{m+1,1:K} \leftarrow (T + 1)K + [0, 1, \ldots, K - 1]$
for $j = m, \ldots, 1$ **do**
  $t_{j,1:K} \leftarrow I_{j+1,\lfloor t_{j+1}/K \rfloor, t_{j+1} \bmod K+1}$
end for
$t \leftarrow \lfloor t/K \rfloor$
Output: Top K segmentations $t_{\cdot,1}, \ldots, t_{\cdot,K}$

of the objective $\mathscr{L}$ is Lipschitz with constant

$$L_{\nabla\mathscr{L}} \coloneqq \frac{4M_\phi L_{\nabla_w \phi}}{\sqrt{T_{\min}}}\left(1+\frac{\lambda}{2\sqrt{\epsilon}}\right) + \frac{M_{\nabla_w\phi}}{\sqrt{T_{\min}}}\sqrt{4+\frac{4\lambda}{\epsilon^{1/2}}+\frac{64\lambda M_\phi^2}{\epsilon^{3/2}}+\frac{\lambda^2}{\epsilon}+\frac{32\lambda^2 M_\phi^2}{\epsilon^2}+\frac{16\lambda^2 M_\phi^4}{\epsilon^3}} + 2\gamma\,,$$

where $T_{\min}$ is the length of the shortest sequence.

From Propositions 11-13 we can then deduce the following result characterizing the convergence of gradient descent on $\mathscr{L}(\{\mathcal{T}^{(i)}\}_{i=1}^n, \cdot; \{x^{(i)}\}_{i=1}^n)$ with a constant step size. The proof follows directly from the analysis in Section 1.2.3 of Nesterov (2004).

Proposition 14. *Denote $\mathscr{L}(w) = \mathscr{L}(\{\mathcal{T}^{(i)}\}_{i=1}^n, w; \{x^{(i)}\}_{i=1}^n)$. Under the assumptions of Propositions 11 and 13 the iterates of gradient descent on $\mathscr{L}(w)$ with constant step size $1/L_{\nabla\mathscr{L}}$ and fixed segmentations $\{\mathcal{T}^{(i)}\}_{i=1}^n$ satisfy*

$$\min_{a=1,\ldots,A} \|\nabla\mathscr{L}(w_a)\|_2^2 \leq \frac{2L_{\nabla\mathscr{L}}\left(\mathscr{L}(w_1)-\mathscr{L}(\hat{w})\right)}{A}\,,$$

where w_a denotes the parameter vector w at iteration a and $\hat{w} = \lim_{a\to\infty} w_a$.

The alternating minimization procedure is guaranteed to converge to a stationary point if $\mathscr{L}(w)$ satisfies the conditions of Proposition 14 and, in addition, has finitely many stationary points. Indeed, no alternating step can increase the objective value, and in this case there are only a finite number of possible objective values that can be obtained after each alternating step.

4.5.2 Optimization via smoothing

The existence of the inner minimization over the change-point parameters $t_1, \ldots, t_{m^{(i)}}$ makes the objective $\mathscr{L}$ non-smooth and difficult to optimize. In the previous subsection we got around this by alternating between updates of the change points and updates of the model

parameters w. In this subsection we will consider smoothing the objective instead in order to be able to apply faster-converging optimization algorithms to a sequence of smooth problems.

We will smooth the problem by smoothing the min function. As smoothing is typically discussed in the context of convex functions, we will convert the min function to a max function in order to derive the smoothed function. We now recall the definition of a smoothable function.

Definition 15 (Smoothable function, Beck and Teboulle, 2012)**.** *Let $g : \mathbb{R}^p \to (-\infty, \infty]$ be a closed and proper convex function and let $\mathcal{X} \subseteq \operatorname{dom} g$ be a closed convex set. The function g is called "(β, γ, K)-smoothable" over $\mathcal{X}$ if there exist γ_1, γ_2 satisfying $\gamma_1 + \gamma_2 = \gamma > 0$ such that for every $\alpha > 0$ there exists a continuously differentiable convex function $g_\alpha : \mathbb{R}^p \to (-\infty, \infty)$ such that the following hold:*

1. $g(x) - \gamma_1 \alpha \leq g_\alpha(x) \leq g(x) + \gamma_2 \alpha \quad \forall\, x \in \mathcal{X}$.

2. *The function g_α has a Lipschitz gradient over $\mathcal{X}$ with Lipschitz constant less than or equal to $K + \beta/\alpha$. That is, there exist $K \geq 0$ and $\beta > 0$ such that*

$$\|\nabla g_\alpha(x) - \nabla g_\alpha(y)\| \leq \left(K + \frac{\beta}{\alpha}\right) \|x - y\| \quad \forall\, x, y \in \mathcal{X}\ .$$

The function g_α is called an "α-smooth approximation" of g over $\mathcal{X}$ with parameters (β, γ, K).

As noted by Beck and Teboulle (2012) and discussed in Appendix C.2, an α-smooth approximation can be derived with any convex and continuously differentiable function $q : \mathbb{R}^p \to \mathbb{R}$ with Lipschitz gradient. In this work we choose to use the ℓ_2^2 smoother given by $\ell_2^2(u) = 0.5(\|u\|_2^2 - 1)$. This particular smoother has two main benefits. First, it requires only computing the gradient of $\mathcal{L}$ for a subset of the set of possible change points. Second, the smoothed version of the max function is always larger than the max function itself (Pillutla et al., 2019).

Smoothed objective and gradient. Consider smoothing the inner min function in problem (4.6) using the ℓ_2^2 smoother. As shown in Appendix C.2, smoothing the function $h(z) \coloneqq \min\{z_1, \ldots, z_p\}$ results in the function

$$h_\alpha(z) = \langle z, \operatorname{proj}_{\Delta^{p-1}}(-z/\alpha)\rangle + \frac{\alpha}{2}\|\operatorname{proj}_{\Delta^{p-1}}(-z/\alpha)\|^2 - \frac{\alpha}{2}\,,$$

where Δ^{p-1} is the $(p-1)$-dimensional probability simplex. Now denote the vector of all possible $\mathcal{L}$'s given w and $x^{(i)}$ by $\mathcal{L}(w; x^{(i)}) = [\mathcal{L}(\mathcal{T}, w; x^{(i)})]_{\mathcal{T}\in\mathscr{T}^{(i)}_{m^{(i)}}}$, where $\mathscr{T}^{(i)}_{m^{(i)}}$ is the set of all possible segmentations of sequence i with $m^{(i)}$ change points. Smoothing $\mathscr{L}$ by smoothing the inner min function results in the problem

$$\min_w \ \frac{1}{n}\sum_{i=1}^{n} h_\alpha(\mathcal{L}(w; x^{(i)})) + \gamma\|w\|_2^2\,. \tag{4.9}$$

We will denote the smoothed objective by $\mathscr{L}_s$. By Proposition 2 of Pillutla et al. (2019), $\mathscr{L}_s$ is bounded below by $\mathscr{L} - \alpha/2$.

Using Theorem 4.1 of Beck and Teboulle (2012), we can compute the gradient of the smoothed objective:

Proposition 16. *The gradient of the smoothed objective $\mathscr{L}_s$ is given by*

$$\nabla_w \mathscr{L}_s = \frac{1}{n}\sum_{i=1}^{n} \left(\nabla_w \mathcal{L}(w; x^{(i)})\right)^T \operatorname{proj}_{\Delta^{\left|\mathscr{T}^{(i)}_{m^{(i)}}\right|-1}}(-\mathcal{L}(w; x^{(i)})/\alpha) + 2\gamma w\,,$$

with

$$\begin{aligned}\nabla_w \mathcal{L}(w; x^{(i)})_{\mathcal{T},\cdot} = \frac{2}{T^{(i)}}\sum_{j=0}^{m^{(i)}}\sum_{t=t_j^{(i)}}^{t_{j+1}^{(i)}-1} \left(\nabla_w \phi_t^{(i)}\right)^T \Bigg\{ &\left(\phi(x_t^{(i)}; w) - \hat{\mu}_j^{(i)}\right) \\ &- \frac{\lambda}{2}\left[\frac{1}{T^{(i)}}\sum_{t=1}^{T^{(i)}} \|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\|_2^2 + \epsilon\right]^{-1/2} \left(\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\right)\Bigg\}\,,\end{aligned}$$

where $\mathcal{T} = \{t_1^{(i)}, \ldots, t_{m^{(i)}}^{(i)}\}$ *is one segmentation of* $x^{(i)}$.

The next proposition shows that this gradient is also Lipschitz. The result follows from Lemma 68, Proposition 69, Theorem 4.1 of Beck and Teboulle (2012), and the fact that the ℓ_2 norm of a vector in the probability simplex is at most 1.

Proposition 17. *Define* $\phi_t^{(i)}(w) = \phi(x_t^{(i)}; w)$ *for all* t, i. *Assume* $\phi_t^{(i)}(\cdot)$ *is differentiable and there exist constants* $M_\phi, M_{\nabla_w \phi} \geq 0$ *such that for all* $x_t^{(i)}$ *and* w *we have* $\|\phi_t^{(i)}(w)\|_2 \leq M_\phi$ *and* $\|\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)}(w))\|_F \leq M_{\nabla_w \phi}$. *Moreover, assume there exists a constant* $L_{\nabla_w \phi} \geq 0$ *such that for all* t *and* i, $\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)})$ *is Lipschitz with constant* $L_{\nabla_w \phi}$. *Then the gradient of the objective* $\mathscr{L}_s$ *is Lipschitz with constant*

$$L_{\nabla_w \mathcal{L}} + \frac{4 M_\phi |\mathscr{T}_{\max}|^{1/2}}{\alpha} \left(1 + \frac{\lambda}{2\sqrt{\epsilon}}\right) + 2\gamma \,,$$

where $L_{\nabla_w \mathcal{L}} = \frac{4}{\sqrt{T_{\min}}} \left(1 + \frac{\lambda}{2\sqrt{\epsilon}}\right) M_\phi L_{\nabla_w \phi} + \frac{M_{\nabla_w \phi}}{\sqrt{T_{\min}}} \sqrt{4 + \frac{4\lambda}{\epsilon^{1/2}} + \frac{64\lambda M_\phi^2}{\epsilon^{3/2}} + \frac{\lambda^2}{\epsilon} + \frac{32\lambda^2 M_\phi^2}{\epsilon^2} + \frac{16\lambda^2 M_\phi^4}{\epsilon^3}}$ *and* $|\mathscr{T}_{\max}| = \max_{i=1,\ldots,n} |\mathscr{T}_{m^{(i)}}^{(i)}|$.

Given this smoothed objective and its gradient, we can apply typical first order methods to optimize $\mathscr{L}_s$. An analogous result to Proposition 14 applies, with the exception that there are no segmentations to fix. If the network ϕ is additionally twice differentiable with respect to w then we can use second-order methods. In our experiments we use gradient descent.

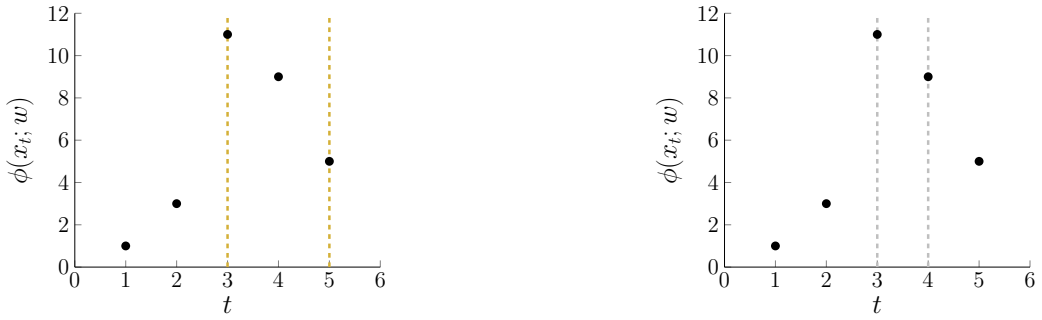
Approximating the smoothed gradient. As written, problem (4.9) results in computationally intractable learning when $|\mathscr{T}_{m^{(i)}}^{(i)}|$ is large because it entails computing all $|\mathscr{T}_{m^{(i)}}^{(i)}|$ elements of $\mathcal{L}(w; x^{(i)})$ for each i. Pillutla et al. (2019), relying on the fact that the projections onto the simplex are sparse, propose a heuristic that entails using a weighted average of the gradients of the K smallest elements of $\mathcal{L}(w; x^{(i)})$ for some integer K. In particular, the approximate gradient when using this heuristic is given by

$$\nabla_w \mathscr{L}_s = \frac{1}{n} \sum_{i=1}^{n} \left(\nabla_w \mathcal{L}(w; x^{(i)})\right)^T \mathcal{E}_K(\mathcal{L}(w; x^{(i)})) \, \mathrm{proj}_{\Delta^{K-1}}(-\mathcal{L}(w; x^{(i)})_{[1:K]}/\alpha) + 2\gamma w \,,$$

where $\mathcal{E}_K(\mathcal{L}(w; x^{(i)})) = [e_{k_1^{(i)}}, \ldots, e_{k_K^{(i)}}]$, $k_\ell^{(i)}$ is the index corresponding to the ℓth smallest value of $\mathcal{L}(w; x^{(i)})$, and $\mathcal{L}(w; x^{(i)})_{[1:K]}$ contains the K smallest values in $\mathcal{L}(w; x^{(i)})$. If the smoothing parameter is sufficiently small, i.e., $\alpha \leq \sum_{k=1}^{K}(\mathcal{L}(w; x^{(i)})_{[K+1]} - \mathcal{L}(w; x^{(i)})_{[k]})$ where $\mathcal{L}(w; x^{(i)})_{[k]}$ is the kth smallest value in $\mathcal{L}(w; x^{(i)})$, then this heuristic is exact (Pillutla et al., 2019).

However, it remains to show how to efficiently compute the K largest entries of $\mathcal{L}(w; x^{(i)})$ for each i. For a fixed w, this amounts to finding the K best solutions to (4.1) for each i. Recall that we can write the objective function in (4.1) as (4.7). From this we can see that to compute the best K segmentations, we need to store the best K segmentations when minimizing over t_1, and then when minimizing over t_2, etc., up through t_m. Storing the best K segmentations can be done efficiently with a max-priority queue, and a max-priority queue can be implemented using a max heap (Williams, 1964; Cormen et al., 2009). A max heap stores elements in a binary tree, such that removing the largest element has computational complexity $O(1)$. When the heap is of size K, inserting an element is $O(\log_2 K)$ and extracting the largest value and updating the heap is $O(\log_2 K)$.

The above reasoning leads to Algorithm 6. In this case, L and I are tensors, with the final dimension indexing the ranking of the segmentation (out of the top K). In addition, the algorithm stores both the change-point locations t_j and the rank k of the previous partial segmentation in I. For $K > 1$ the computational complexity of the forward pass of the algorithm is $O(mT^2K \log_2 K)$ and that of the backward pass is $O(mK)$. Therefore, the computational complexity of the entire algorithm is $O(mT^2K \log_2 K)$. The storage cost is $O(mTK)$. Figure 4.3 demonstrates how the change-point estimation works on data corresponding to Figure 4.2 in the case where $K = 2$ and one change point (at $t = 3$) is known. The gold boxes represent the path leading to the best segmentation, while the silver boxes represent the path leading to the second-best segmentation.



(a) Example dataset $\phi(x; w) = [1, 3, 11, 9, 5]$ with the best set of change points displayed in gold (left) and the second-best set displayed in silver (right).

$L_{\cdot,\cdot,1} =$

∞	0	2	∞	∞	∞	1
∞	∞	0	2	4	∞	2
∞	∞	∞	∞	∞	4	3
1	2	3	4	5	6	

Location of next change point, t_{j+1}

Next change-point number, $j+1$

$L_{\cdot,\cdot,2} =$

∞	∞	∞	∞	∞	∞	1
∞	∞	∞	∞	∞	∞	2
∞	∞	∞	∞	∞	26.5	3
1	2	3	4	5	6	

Location of next change point, t_{j+1}

Next change-point number, $j+1$

(b) Best partial objectives (left) and second-best partial objectives (right) corresponding to the data in (a). The gold boxes show the path to the optimum objective value while the silver boxes show the path to the second-best objective value.

$I_{\cdot,\cdot,1} =$

-	1	1	-	-	-	1
-	-	4	6	6	-	2
-	-	-	-	-	10	3
1	2	3	4	5	6	

Location of next change point, t_{j+1}

Next change-point number, $j+1$

$I_{\cdot,\cdot,2} =$

-	-	-	-	-	-	1
-	-	-	-	-	-	2
-	-	-	-	-	8	3
1	2	3	4	5	6	

Location of next change point, t_{j+1}

Next change-point number, $j+1$

(c) $t_j K + k$, where t_j and k are used to index the best partial objectives (left) and second-best partial objectives (right) corresponding to the data in (a).

Figure 4.3: Example data, estimated change points, and tensors L and I from data corresponding to Figure 4.2 when $K = 2$ and the change point at index $t = 3$ is known.

4.5.3 Optimization error

Two sources of error can exist when optimizing the objective $\mathscr{L}$ or $\mathscr{L}_s$. First, since the problem is non-convex, the optimization algorithm may not reach the global optimum. Second, if using the smoothed objective $\mathscr{L}_s$, its optimum for the parameters w may not be the same as that of $\mathscr{L}$. In the remainder of this section we bound the amount of error introduced by these two sources.

Error from optimizing the non-smooth objective. We begin by introducing some additional notation. Define $w^\star$ to be a w yielding a global minimum of (4.6) and define $\hat{w}$ to be a w returned by optimizing (4.6). Moreover, for a generic sequence x, define $\hat{\mathcal{T}} \in \arg\min_{\mathcal{T}\in\mathcal{C}} \mathscr{L}(\mathcal{T}, \hat{w}; x)$ and $\mathcal{T}^\star \in \arg\min_{\mathcal{T}\in\mathcal{C}} \mathscr{L}(\mathcal{T}, w^\star; x)$. There are numerous ways in which one can measure the distance or dissimilarity between two sets of change-point estimates. We will prove a bound in terms of the distance between normalized equivalence matrices describing the segmentations. Define the normalized segment equivalence matrix $\Pi_{\mathcal{T}} \in \mathbb{R}^{T\times T}$ by

$$[\Pi_{\mathcal{T}}]_{t,t'} = \begin{cases} \frac{1}{t_j - t_{j-1}}, & \text{if } t_{j-1} \le t, t' < t_j \text{ for some } j \\ 0, & \text{else .} \end{cases}$$

We then have the following upper and lower bounds on the Frobenius distance $d_F(\mathcal{T}, \mathcal{T}') = \|\Pi_{\mathcal{T}} - \Pi_{\mathcal{T}'}\|_F$ between segmentations. The proof may be found in Appendix C.3.

Proposition 18. *A lower bound on the Frobenius norm distance between the segmentations $\hat{\mathcal{T}}$ and $\mathcal{T}^\star$ is given by*

$$d_F(\hat{\mathcal{T}}, \mathcal{T}^\star) \ge \frac{1}{\|\Phi(x;\hat{w})\|_F} \left| \|\Phi(x;\hat{w}) - \Pi_{\hat{\mathcal{T}}}\Phi(x;\hat{w})\|_F - \|\Phi(x;\hat{w}) - \Pi_{\mathcal{T}^\star}\Phi(x;\hat{w})\|_F \right| .$$

Furthermore, if the smallest eigenvalue of the normalized Gram matrix $\Phi(x;\hat{w})\Phi(x;\hat{w})^T/T$,

$\lambda_{\min}$, *satisfies* $\lambda_{\min} > 0$, *then*

$$d_F(\hat{\mathcal{T}}, \mathcal{T}^\star) \leq \frac{2}{\lambda_{\min} T} \|\Phi(x; \hat{w})\|_F \|\Phi(x; \hat{w}) - \Pi_{\mathcal{T}^\star} \Phi(x; \hat{w})\|_F \ .$$

It is also possible to prove bounds on the maximum distance between change points in $\hat{\mathcal{T}}$ and $\mathcal{T}^\star$ (i.e., in terms of the Hausdorff distance). This can be done using the relationships between the Frobenius norm distance and the Hausdorff distance (Lajugie et al., 2014; Garreau and Arlot, 2018).

Error from optimizing the smoothed objective. Using the smoothed version of the objective, $\mathscr{L}_s$, introduces error that depends on the parameter α. The following proposition bounds the difference in the objective value $\mathscr{L}(\tilde{\mathcal{T}}, \tilde{w}; x)$ obtained when using parameters $\tilde{w}$ optimizing the smoothed objective and the objective value $\mathscr{L}(\mathcal{T}^\star, w^\star; x)$ obtained when using the parameters $w^\star$ optimizing the non-smooth objective.

Proposition 19. *Define* $\tilde{w}$ *to be an optimizer of the smoothed objective, i.e.,*

$$\tilde{w} \in \arg\min_w \frac{1}{n} \sum_{i=1}^{n} h_\alpha(\mathcal{L}(w; x^{(i)})) + \gamma \|w\|_2^2 \ .$$

Furthermore, define $w^\star$ *to be an optimizer of the non-smooth objective, i.e.,*

$$w^\star \in \arg\min_w \frac{1}{n} \sum_{i=1}^{n} h_0(\mathcal{L}(w; x^{(i)})) + \gamma \|w\|_2^2 \ ,$$

where $h_0(\mathcal{L}(w; x^{(i)})) \coloneqq \min_{\mathcal{T}^{(i)} \in \mathcal{C}^{(i)}(m^{(i)})} \mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)})$. *Then for all* x *we have*

$$\begin{aligned} &\frac{1}{n} \sum_{i=1}^{n} h_\alpha(\mathcal{L}(\tilde{w}; x^{(i)})) - \frac{1}{n} \sum_{i=1}^{n} h_0(\mathcal{L}(\tilde{w}; x^{(i)})) \\ \leq &\left| \left(\frac{1}{n} \sum_{i=1}^{n} h_0(\mathcal{L}(\tilde{w}; x^{(i)})) + \gamma \|\tilde{w}\|_2^2 \right) - \left(\frac{1}{n} \sum_{i=1}^{n} h_0(\mathcal{L}(w^\star; x^{(i)})) + \gamma \|w^\star\|_2^2 \right) \right| \end{aligned}$$

$$\leq \frac{1}{n}\sum_{i=1}^{n} h_\alpha(\mathcal{L}(\tilde{w}; x^{(i)})) - \frac{1}{n}\sum_{i=1}^{n} h_0(\mathcal{L}(w^\star; x^{(i)})) + \frac{1}{2}\alpha + \gamma\|\tilde{w}\|_2^2 - \gamma\|w^\star\|_2^2 \,.$$

Proof. For the lower bound we have

$$\begin{aligned}
&\left|\left(\frac{1}{n}\sum_{i=1}^{n} h_0(\mathcal{L}(\tilde{w}; x^{(i)})) + \gamma\|\tilde{w}\|_2^2\right) - \left(\frac{1}{n}\sum_{i=1}^{n} h_0(\mathcal{L}(w^\star; x^{(i)})) + \gamma\|w^\star\|_2^2\right)\right| \\
&\geq \frac{1}{n}\sum_{i=1}^{n} h_\alpha(\mathcal{L}(\tilde{w}; x^{(i)})) + \gamma\|\tilde{w}\|_2^2 - \frac{1}{n}\sum_{i=1}^{n} h_0(\mathcal{L}(w^\star; x^{(i)})) - \gamma\|w^\star\|_2^2 \\
&\geq \frac{1}{n}\sum_{i=1}^{n} h_\alpha(\mathcal{L}(\tilde{w}; x^{(i)})) - \frac{1}{n}\sum_{i=1}^{n} h_0(\mathcal{L}(\tilde{w}; x^{(i)}))
\end{aligned}$$

by the definitions of $w^\star$ and $\tilde{w}$ and Proposition 2 of Pillutla et al. (2019). The upper bound follows from Proposition 2 of Pillutla et al. (2019). □

4.6 Experiments

We illustrate our approach on sequences of real-valued data and images, and investigate the benefit of learning with labeled change points. Given a parameterized feature representation mapping, the approach can potentially be applied to any kind of data and can leverage any number of labeled change points. In this section we address the following questions:

1. Does learning a feature representation (with or without labeled data) result in improved estimation of the change points?

2. Does smoothing the objective lead to improved feature representations by better optimizing the objective (4.6)?

4.6.1 Experimental details

Next we describe the synthetic and real-world data used in the experiments, along with the corresponding networks used to learn feature representations.

Datasets and architectures. In the experiments we focus on three datasets: (1) Simulated data; (2) MNIST-seq, a dataset we created from MNIST (LeCun et al., 2001); and (3) the Bee Waggle Dance dataset (Oh et al., 2008).

The simulated data consists of sequences of length 100. It is created by alternating between series of draws from $N(0, 2^{-4})$ and $N(1, 2^{-4})$ that are then passed through the two-layer RBF network from Wu et al. (2019) with parameters randomly drawn from $N(0, 9)$. The number of change points in each sequence is selected uniformly at random between 1 and 10. The locations of the change points are chosen sequentially and uniformly at random subject to the constraint that all change points are at least 5 observations apart. There are 500 training sequences, 100 validation sequences, and 100 test sequences. MNIST-seq consists of sequences of 100 MNIST images of dimension 28×28. The number and locations of the change points (changes in the labels of the images) are selected in the same manner as for the synthetic data. There are approximately 500 training sequences, 100 validation sequences, and 100 test sequences for a given random seed. Finally, the Bee Waggle Dance dataset contains sequences of position and angle recordings of individual bees performing a "waggle dance". Such a dance is used to indicate to other bees the location of a potential food source (von Frisch, 1993). It consists of three actions: turn left, waggle, and turn right. The sequences have lengths ranging from 602 to 1124 and have dimension 3. Each sequence has between 15 and 28 change points (changes in action). Following Chang et al. (2019), we divide the data into four training sequences, one validation sequence, and one test sequence.

The datasets were transformed as follows. The synthetic data was unchanged. The images in MNIST-seq were standardized using the mean and standard deviation across all pixels in the combined training and validation sets. The Bee Waggle Dance data was transformed as done by Chang et al. (2019). Namely, the angle θ was transformed into the difference between $\cos(\theta)$ and $\sin(\theta)$. Moreover, each feature was scaled to lie in $[0, 1]$. Then, as there is autocorrelation in the Bee Waggle Dance data, we considered differencing the data and concatenating features within sliding windows. Based on the validation results we use first differences and sliding windows of size 3. For more details on the datasets, see Appendix C.4.

The architectures we use are the true architecture for the synthetic dataset (but initialized randomly) and kernel networks with 32 filters per layer for the real datasets. Each layer ℓ of such a kernel network approximates a kernel k_ℓ using the Nyström features $\phi_\ell(x) = (k_\ell(w_\ell^T w_\ell) + \epsilon I)^{-1/2} k_\ell(w_\ell^T x)$ with learnable parameters w_ℓ and regularization $\epsilon > 0$ (Williams and Seeger, 2000; Mairal, 2016). For the Bee Waggle data we use five-layer kernel networks. For MNIST-seq we use the convolutional kernel network (CKN) version of LeNet-5 (LeCun et al., 2001) proposed in Chapter 2.

Training. The initialization and hold-out validation for each dataset/architecture is as follows. The initial weights of the RBF network are drawn from $N(0, 1)$ while the initial weights of the kernel networks are set using randomly-sampled feature representations from each layer of the network. The bandwidths of the kernels, the regularization parameters ϵ, λ, and γ (if applicable), and the learning rate are set using hold-out validation. The value of γ is set to zero for the kernel networks because the filters are already constrained to lie on a product of spheres. In each case the minimum distance Δ between change points is set to 1. The hold-out validation was performed for the case of zero labeled sequences. The resultant hyperparameter values were then used for all other settings. After performing hold-out validation we do not retrain on the combined training and validation sets.

When training, the optimization is performed for 100 iterations and uses one step of (projected) gradient descent to update w at each iteration. Due to memory constraints we use a mini-batch size of 50 sequences for the LeNet-5 CKN on MNIST-seq. We report the test results at the iteration where the smallest Frobenius distance was observed on the validation set. When not specified, we use the alternating version of the optimization rather than the smoothed version. In the smoothing experiments we set the parameter α based on the training objective vs. iteration curve generated with a single seed. For more details regarding the training, see Appendix C.5.

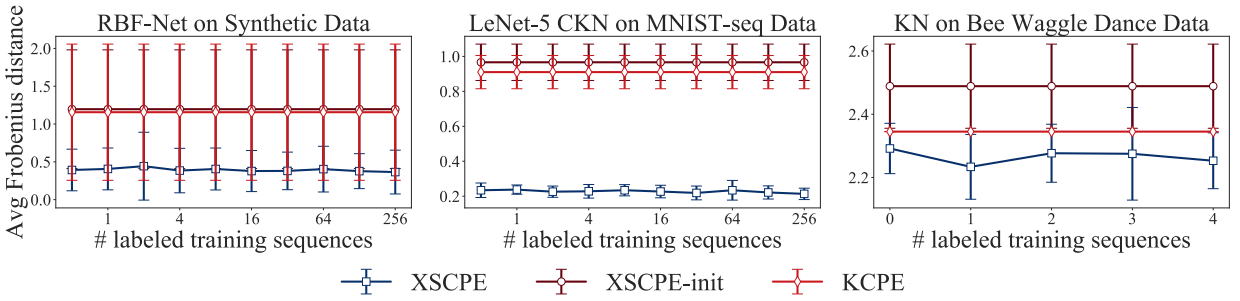


Figure 4.4: Average Frobenius distance between the true and estimated change points on the test sets when varying the number of labeled training sequences. The error bars represent one standard deviation across 10 trials with different random seeds. XSCPE is the algorithm proposed in this chapter.

Code. The code for the experiments uses Faiss, PyTorch, and the YesWeCKN code from Chapter 2 (Johnson et al., 2019; Paszke et al., 2019). The code may be found online at `https://github.com/cjones6/chpt-learn`.

4.6.2 Results

Comparison to baselines. Figure 4.4 presents results on each dataset, comparing the average Frobenius distance across three methods when the number of labeled training sequences varies. We compare our method, XSCPE, to optimizing (4.1) on the network outputs at initialization (XSCPE-init) and to applying the kernel change-point method of Harchaoui and Cappé (2007) with a Gaussian RBF kernel whose bandwidth was tuned on the validation set (KCPE).

There are three main takeaways from Figure 4.4. First, learning the feature representation improves upon using the features from the untrained network. On MNIST-seq learning the feature representation decreased the average Frobenius distance by a median of 77% across all numbers of labeled training sequences and trials. On the synthetic data and Bee Waggle data these values were 56% and 11%, respectively. Second, learning the feature representation can result in a better performance than with KCPE. The gain on MNIST-seq was the largest, which saw a median improvement of 75%. The gains on the synthetic data and Bee Waggle

data were lower, at 52% and 2%, respectively. This suggests that the method is most effective when a network is used that takes advantage of structure in the data (as is the case for MNIST-seq) and is less effective when the amount of training data is small (as is the case for the Bee Waggle data). Finally, it is noteworthy that the number of labeled training sequences makes less difference. The average Frobenius distance decreased by a median of 6% on MNIST-seq when going from 0 to 256 labeled sequences. In contrast, on the synthetic data it increased by 3%. On the Bee Waggle data the median decrease was 1% when going from 0 to 4 labeled sequences.

Effects of learning. Figure 4.5 shows the effects of training the LeNet-5 CKN on MNIST-seq with no labeled sequences. The left plot shows that the objective consistently decreased during the first 75 iterations. From the middle plot we can see that for the given sequence, three change points were initially incorrectly estimated but after 77 iterations all were estimated correctly. Finally, the right plots display the t-SNE representations (Van Der Maaten and Hinton, 2008) of the features for the MNIST digits before and after training. They suggest that XSCPE learns representations that can be used to better cluster the digits. This is especially true of the digits 6 (dark green) and 4, 7, and 9 (lime green, light blue, and purple).

Sensitivity to parameter values. Figure C.7 in Appendix C.6 shows the sensitivity of the results on the test set of MNIST-seq when training a LeNet-5 CKN with no labeled sequences for 100 iterations. As expected, the algorithm struggles to train when the step size is too large (greater than 2^{-3}) and converges slowly if it is too small (smaller than 2^{-7}). However, perhaps surprisingly, the algorithm is quite sensitive to the value of λ. In particular, only $\lambda = 4, 8$, and 16 are within a factor of two of the optimal average Frobenius distance. On the other hand, the algorithm is insensitive to ϵ as long as ϵ is sufficiently small (at most 10). Finally, the Frobenius distance improves by a median of 82% as the number of training sequences increases from 1 to 512.

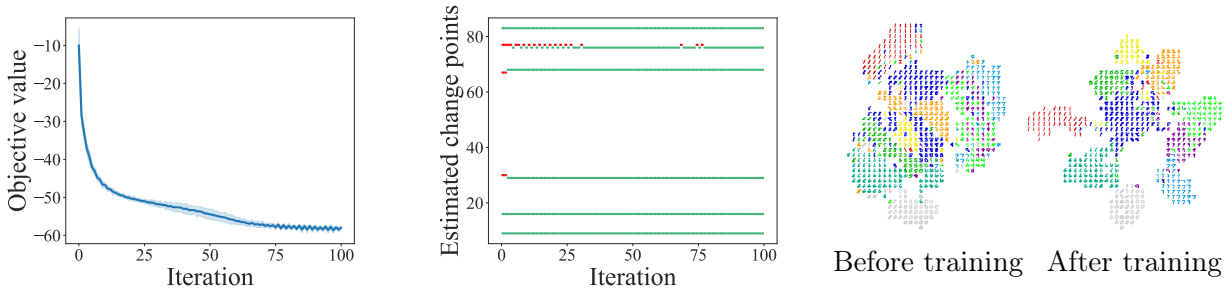


Figure 4.5: Left: Average objective value on the training set when training the LeNet-5 CKN on MNIST-seq with no labeled sequences. The error band represents one standard deviation from the mean. Center: Estimated change points for one sequence from MNIST-seq. Green and red indicate correctly and incorrectly estimated change points, respectively. Right: t-SNE visualization of MNIST-seq features with the random initial network (left) and after training with XSCPE (right). The colors represent the true labels. Zoom in to see individual digits.

Effect of smoothing. Thus far the optimization has been performed via the alternating approach. Now we will examine the effect of the smoothing approach. Figure 4.6 plots the average objective value vs. iteration across 10 trials with different random seeds. Each curve corresponds to one value of K used in the top-K heuristic. From the figure we can see that the training on the synthetic data was rather jagged. While at times smoothing with, e.g., $K = 10$ outperformed not smoothing, this was mainly the case after approximately 65 iterations. Moreover, examining the curves for individual seeds, we did not find a discernible pattern regarding when one value of K outperformed the others. In contrast, smoothing only hurts when training the LeNet-5 CKN on MNIST-seq and makes no noticeable difference when training the kernel network on the Bee Waggle data. Figure C.8 in Appendix C.6 shows the effect of decaying α across iterations. The general conclusions from this case are the same. Finally, Figures C.9 and C.10 in Appendix C.6 display the average number of non-zero entries in $\text{proj}_{\Delta^{K-1}}(-\mathcal{L}(w; x^{(i)})_{[1:K]}/\alpha)$ at each iteration. From the plots we can see that the number of non-zeros tends to decrease as the objective value decreases, and more so in the decaying α case, as expected.

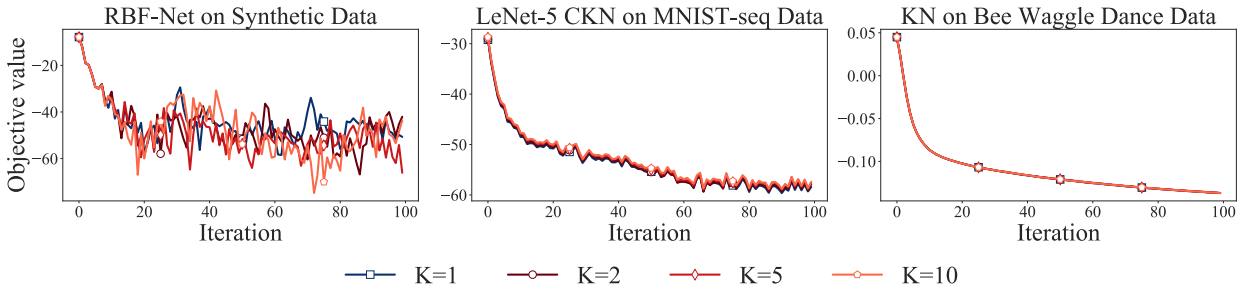


Figure 4.6: Average objective value on the training sets when varying the value of K in the top-K heuristic. The objective values are averaged across 10 trials with different random seeds. The smoothing parameter α was fixed across iterations to 10, 1000, and 1, respectively, for the simulated, MNIST-seq, and Bee Dance experiments.

4.7 Conclusion

In this chapter we proposed a framework for learning feature representations for retrospective change-point estimation that can leverage information provided by labeled change points. We examined two methods for optimizing this objective, based on alternating optimization and smoothed optimization. With experiments on both synthetic and real-world data we demonstrated that learning the feature representations can lead to improved segmentations, regardless of the number of labeled change points. Throughout the chapter we assumed the number of change points is known. We intend to pursue the idea of learning the number of change points in future work.

Chapter 5

MAPPING MARINE PHYTOPLANKTON BIOGEOGRAPHY USING KERNEL-BASED CHANGE-POINT ESTIMATION

Joint work with S. Clayton, F. Ribalet, E. V. Armbrust, and Z. Harchaoui.[1]

Abstract. Automated, ship-board flow cytometers provide high-resolution maps of phytoplankton communities over large swaths of the world's oceans and thus pave the way for understanding how environmental conditions shape community structure. Identification of community changes along a cruise transect commonly segment the data. However, existing segmentation methods are not applicable to flow cytometry data as these data are stored as "point cloud" data, with each data file containing thousands of particle measurements. Here, we describe a kernel-based change-point estimation method for point cloud data that can readily scale to hundreds of millions of particle measurements per cruise. We provide a data-driven method for estimating the number of change points by fitting parameters on separate but related time series of physical data collected simultaneously with the flow cytometry data. This method uses the particle measurements and does not require prior clustering of particles to define taxa labels, eliminating a potential source of error. Our change-point method successfully locates changes in phytoplankton community structure and represents an important advance in automating the analysis of such large datasets now emerging in biological oceanography.

---

[1]This work was first presented at the Ocean Sciences Meeting in February 2018. We gratefully acknowledge support from the Canadian Institute for Advanced Research, the Simons Foundation, the Washington Research Foundation, the Moore-Sloan Data Science Environments, and faculty research awards.

5.1 Introduction

Phytoplankton, microscopic photosynthetic organisms living near the surface of bodies of water, play two major roles on Earth. First, they transfer carbon dioxide from the atmosphere to the ocean. Studies suggest that phytoplankton account for approximately half of the photosynthesis that occurs on Earth (Field et al., 1998). Second, phytoplankton form the base of the oceanic food chain. As such, changes in the abundance of phytoplankton species both affect and are affected by global warming (Falkowski et al., 1998). On the one hand, an increased abundance of phytoplankton leads to more carbon being removed from the atmosphere. This carbon is transferred deeper into the ocean as phytoplankton die and are consumed by other creatures. On the other hand, as global warming increases surface ocean temperatures, the water column becomes more stratified, thereby reducing the vertical flux of nutrients into the surface ocean. It is thought that this might result in decreased oceanic primary production in the future ocean, resulting in reduced carbon drawdown as well as a reduction in food available for higher trophic levels. Therefore, understanding how phytoplankton communities vary in time and space across ocean basins is critical for predicting how marine ecosystems will respond to future climate change.

For the past three decades, flow cytometry has been instrumental to studying the distribution of phytoplankton communities (Sosik et al., 2010). Flow cytometry measures light scatter and fluorescence emissions of individual cells at rates of up to thousands of cells per second. Light scattering is proportional to cell size, and fluorescence is unique to the emission spectra of pigments; these parameters can be used to identify populations of phytoplankton with similar optical properties. Automated flow cytometers such as CytoBuoy (Dubelaar et al., 1999), FlowCytoBot (Olson et al., 2003), and SeaFlow (Swalwell et al., 2011) have provided unprecedented views of dynamics of phytoplankton across large temporal and spatial scales. The recent release of SeaFlow data collected underway during 60 cruises conducted in the North Pacific Ocean (Figure 5.1) offers unique opportunities to study how phytoplankton communities vary over space and time (Ribalet et al., 2019).

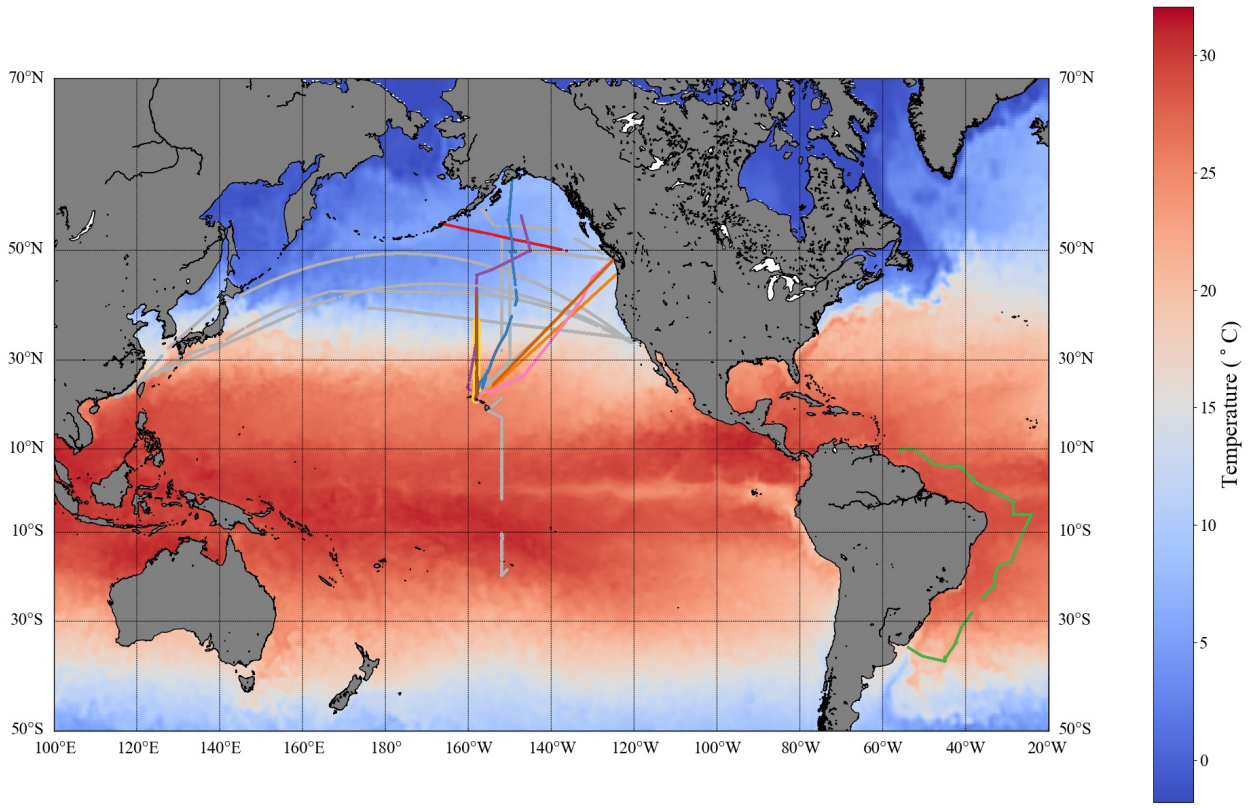
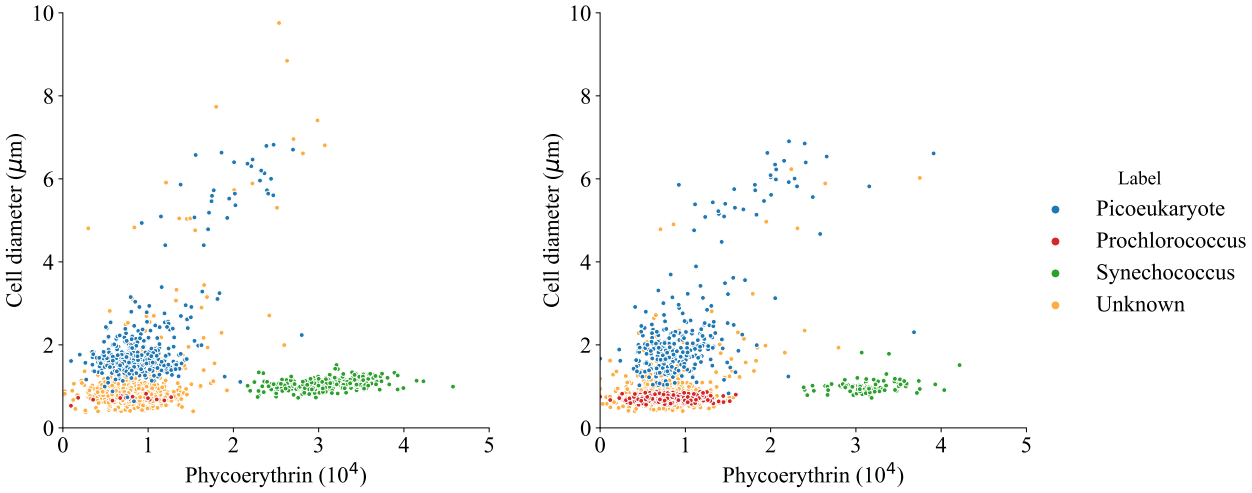


Figure 5.1: Locations of the cruises analyzed in this chapter, overlaid on sea surface temperature data from April 26, 2016.[2] Only physical data was used from the cruises in gray. In this work we primarily focus on KOK1606, the cruise in bright yellow in the middle of the map, which took place from April 20-May 4, 2016.

A conventional approach to understanding community structure along a cruise transect is to segment the data. However, the underway flow cytometry data produced by instruments such as SeaFlow presents novel challenges not addressed by existing segmentation methods. The individual phytoplankton measurements are stored in files representing three-minute windows (roughly a 1km spatial resolution). Hence, each observation can be viewed as a point cloud of individual phytoplankton measurements. Moreover, as the datasets collected during any given research cruise might contain more than 100 million particle measurements, the

[2]NOAA_OI_SST_V2 data provided by the NOAA/OAR/ESRL PSL, Boulder, Colorado, USA, from their website at `https://psl.noaa.gov/`



(a) Distribution at 2170km

(b) Distribution at 2180km

Figure 5.2: Distribution of the cell diameter and phycoerythrin measurements of phytoplankton 5 kilometers before and after an estimated change point at 2175km along the cruise track of KOK1606 (see Table D.1 in Appendix D.1).

method must scale. Prior methods for change-point estimation in ecology and oceanography apply to real-valued and vectorial data (e.g., Quandt, 1958; Hamilton, 1990; Matteson and James, 2014). Applying such methods by reducing each point cloud to its mean as done by Hyrkas et al. (2015) results in the loss of important information regarding the distribution of the data in each point cloud. An example of a change in distribution of phytoplankton measurements is depicted in Figure 5.2.

In this work we develop a nonparametric statistical approach for identifying abrupt changes in large-scale time series data consisting of point clouds and use it to segment the SeaFlow data along cruise tracks. Our approach first implicitly maps each point cloud to an empirical mean element in a reproducing kernel Hilbert space (Schölkopf and Smola, 2002). Afterward, using the kernel trick, it estimates the locations of abrupt changes in these empirical mean elements along the time series. As computing a kernel on millions of individual points from the point clouds is computationally intractable, we use approximate feature maps produced

via the Nyström method (Williams and Seeger, 2000). We estimate the number of change points by fitting the parameter of the penalty proposed by Lebarbier (2005) on separate but related time series.

When applied to the SeaFlow data, our approach offers both a potential time savings in analyzing the data, and the opportunity to study new scientific questions. One main benefit of the method is that it relies on readily available particle measurements and not species labels. The latter are often obtained via manual annotation, which is labor intensive, and are prone to errors (Hyrkas et al., 2015). We also highlight in our experiments how the estimated change points can be used to address a long-standing scientific question of whether biological shifts coincide with physical shifts in the ocean environment.

5.2 Related Work

The goal of this work is to determine the number and locations of abrupt changes in time series data of phytoplankton communities. There are three general approaches to doing so: (1) repeatedly perform hypothesis tests at every location in the time series and set the testing threshold to yield an appropriate number of change points; (2) estimate the locations of a large number of potential change points and then prune them using a penalty term or hypothesis testing; and (3) fit a single model to the time series that allows for an unknown number of changes at unknown times. The first two approaches are the most straightforward in our setting.

In this section we briefly review change-point methods that have been applied in ecology and oceanography. The identification of abrupt changes goes by a variety of names, including change-point detection, change-point estimation, chronological clustering, regime shift analysis, and temporal segmentation. In this work we will use the the term "change-point estimation" to refer to methods that estimate locations of possible changes. In addition, we will use the term "change-point analysis" to refer to methods that estimate the number of changes and their locations.

Repeated hypothesis testing. Most change-point analysis methods used in ecology and oceanography are based on hypothesis tests performed at each location in a sequence. These methods can be divided into parametric and nonparametric approaches. Page (1954) introduced the CUSUM method, which is an online algorithm based on the sequential probability ratio test. It iterates over a sequence of observations, keeping a running total of the log-likelihood ratio that is reset to zero whenever it drops below zero. When this running total rises above a pre-determined threshold, a change point is detected. Quandt (1958, 1960) proposed an alternative offline procedure for detecting a single change in mean or slope. Specifically, at every time point, two linear regression models are fit: one on data before the time point and one on data after the time point. To test the significance of a change, Quandt proposed using a t-test or an F-test based on the residuals. Later, Srivastava and Worsley (1986) proposed a greedy offline binary segmentation procedure based on the likelihood ratio test for normally distributed data. It first performs the likelihood ratio test at every time point. If the largest such test statistic is significant, the corresponding location is considered to be a change point. The testing then resumes on the segments before and after this change point in an effort to locate additional change points. The process continues until no more significant changes are found.

There are two nonparametric methods that have also been used. The analysis of similarities (ANOSIM) method of Clarke (1993) is quite popular in ecology. ANOSIM works on a dissimilarity matrix between all pairs of observations. To detect a single change the pairs of dissimilarities are converted to rankings. A test statistic is then computed based on the scaled difference between the average rank across and within groups. A p-value can be obtained via a permutation test. More recently, Matteson and James (2014) proposed a nonparametric approach for finding a single change point. It computes the maximum of a sum of U-statistics and a sample average based on observations before and after potential change points. To compute multiple change points the authors suggest performing binary segmentation and testing for the significance of each potential change point using a permutation test.

Estimation and pruning. One method used in oceanography relies on first estimating change points and then determining which ones are relevant. This is the constrained clustering method of Gordon and Birks (1972). In this method a dissimilarity matrix is first computed between all pairs of observations. Next, constrained agglomerative or divisive clustering is applied. Finally, the number of change points is determined by examining when a measure of the variability within segments levels off.

Simultaneous model fitting. Finally, several methods have been used that simultaneously fit a model and determine the number and locations of change points. One method is the Markov-switching vector autoregression approach of Hamilton (1990). This method, extending the independent and identically distributed (i.i.d.) approach of Goldfeld and Quandt (1973), assumes that there exists a fixed number of possible "regimes" in which the observations lie. It treats the regime number as a hidden variable and estimates the hidden states in addition to a parameter vector. Fearnhead (2006) takes an alternative, Bayesian, approach that considers a likelihood based on the assumption that both the observations within each segment and the parameters across segments are independent. He proposes putting priors on the model parameters, the number of change points, and the positions of the change points given the number of change points. Inference can then be performed by sampling from the posterior.

A wide range of additional methods, including many nonparametric approaches, have been proposed and used outside of ecology and oceanography (Basseville and Nikiforov, 1993; Brodsky and Darkhovsky, 1993; Kay, 1993; Tartakovsky et al., 2015; Truong et al., 2020). Of the methods discussed in this section only the dissimilarity-based approaches of Clarke (1993) and Gordon and Birks (1972) could potentially be applied to data consisting of sequences of point clouds. Even if these methods were used, an appropriate dissimilarity measure would still need to be chosen and a principled method for determining the number of changes would need to be proposed. We take a different nonparametric approach, which we motivate and describe in the next section.

5.3 Change-Point Analysis on Point Cloud Data

In this section we present our approach to change-point analysis on sequences of point clouds. To set up the problem, consider an ordered sequence of point clouds $x_1, \ldots, x_T$. In the present context, for every time t, the point cloud $x_t = \{x_{t,1}, \ldots, x_{t,n_t}\}$ consists of n_t points $x_{t,i} \in \mathbb{R}^d$. Assume that the measurements in each point cloud x_t are realizations of random variables drawn from an unknown probability distribution $\mathbb{P}_t$. We seek to estimate the times at which the distributions $\mathbb{P}_t$ change in order to divide the sequence of point clouds into homogeneous segments. That is, if there are m change points, we seek to find the times $t_0 \coloneqq 1 < t_1 < t_2 < \cdots < t_m < t_{m+1} \coloneqq T + 1$ for which $\mathbb{P}_{t-1} \neq \mathbb{P}_t$.

As mentioned in the previous section, one can perform change-point analysis on time series of point clouds by either (1) performing repeated hypothesis testing; or (2) estimating the locations of a large number of potential change points and then using a penalty to estimate which set of possible change points is correct. In each case one could take either a parametric or nonparametric approach. A parametric approach would consist of fitting, e.g., mixture models to the point clouds and then identifying changes in the parameters. The problem with this approach is that model specification could be difficult and model misspecification can negatively impact the consistency and error rates (Song et al., 2016; Aston and Kirch, 2018). Alternatively, a nonparametric approach could consist, for example, of computing density estimates for each point cloud and then identifying significant changes in the density estimates. On the surface, it may appear as though this latter approach would not scale and would perform poorly for high-dimensional data. However, we will see that these potential flaws can be avoided.

In the remainder of this section we first review the relationship between kernel density estimation and the maximum mean discrepancy (Gretton et al., 2012). We then describe several ways in which the maximum mean discrepancy can be used for change-point estimation and present our approach.

5.3.1 Distances between density estimates

Consider two point clouds, x_s and x_t, consisting of i.i.d. observations from distributions $\mathbb{P}_s$ and $\mathbb{P}_t$, respectively. Assume that $\mathbb{P}_s$ and $\mathbb{P}_t$ are absolutely continuous with densities f_s and f_t, respectively. Given a function $\kappa : \mathbb{R}^d \to \mathbb{R}$ satisfying $\kappa \geq 0$ and $\int \kappa(x)dx = 1$, along with bandwidths h_{s,n_s} and h_{t,n_t}, we can form the kernel density estimates

$$\hat{f}_{n_s}(x) = \frac{1}{n_s h_{s,n_s}^d} \sum_{i=1}^{n_s} \kappa\left(\frac{x - x_{s,i}}{h_{s,n_s}}\right) \quad \text{and} \quad \hat{f}_{n_t}(x) = \frac{1}{n_t h_{t,n_t}^d} \sum_{i=1}^{n_t} \kappa\left(\frac{x - x_{t,i}}{h_{t,n_t}}\right)$$

(Rosenblatt, 1956; Parzen, 1962). The kernel density estimators are consistent at every continuity point of the densities f_t and f_s as long as κ is a Borel function satisfying $\sup_{x\in\mathbb{R}^d} |\kappa(x)| < \infty$, $\int |\kappa(x)|dx < \infty$, and $\lim_{\|x\|\to\infty} \|x\|^d \kappa(x) = 0$, and for each $n \in \{n_t, n_s\}$ we have $\lim_{n\to\infty} h_{n_s,n_t} = 0$ and $\lim_{n\to\infty} n h_{n_s,n_t}^d = \infty$ (Cacoullos, 1966).

To test whether the densities f_s and f_t are the same, we can consider using the squared L_2 distance between them (Anderson et al., 1994). Taking $h_{s,n_s} = h_{t,n_t} =: h_{n_s,n_t}$ in order to avoid estimating additional terms, we find

$$\begin{aligned} D_2(\hat{f}_{n_s}, \hat{f}_{n_t})^2 &= \int \left(\hat{f}_{n_s}(x) - \hat{f}_{n_t}(x)\right)^2 dx \\ &= \int \left[\frac{1}{n_s h_{n_s,n_t}^d} \sum_{i=1}^{n_s} \kappa\left(\frac{x - x_{s,i}}{h_{n_s,n_t}}\right) - \frac{1}{n_t h_{n_s,n_t}^d} \sum_{i=1}^{n_t} \kappa\left(\frac{x - x_{t,i}}{h_{n_s,n_t}}\right)\right]^2 dx \\ &= \frac{1}{n_s^2} \sum_{i,j=1}^{n_s} k_x(x_{s,i}, x_{s,j}) - \frac{2}{n_s n_t} \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} k_x(x_{s,i}, x_{t,j}) + \frac{1}{n_t^2} \sum_{i,j=1}^{n_t} k_x(x_{t,i}, x_{t,j}) \ , \quad (5.1) \end{aligned}$$

where $k_x(x_{s,i}, x_{s,j}) = h_{n_s,n_t}^{-2d} \int \kappa\left(\frac{x-x_{s,i}}{h_{n_s,n_t}}\right) \kappa\left(\frac{x-x_{s,j}}{h_{n_s,n_t}}\right) dx$. If k_x is computable in closed form then we can in fact avoid explicitly performing density estimation. Anderson et al. (1994) studied the power of the test under local alternatives and found that the power is larger when the bandwidth is fixed, i.e., when the kernel density estimators are inconsistent.

Gretton et al. (2012) shed light on this enigma by generalizing the test and viewing it from the kernel perspective. Observe that k_x is a kernel, i.e., a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for

some non-empty set $\mathcal{X}$ such that there exists a Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \to \mathcal{H}$ such that for all $x, x' \in \mathcal{X}$, $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$ (Schölkopf and Smola, 2002). Therefore, one could consider letting k_x be a different kernel. Indeed, for a general kernel k the distance (5.1) can be derived using the concept of kernel mean embeddings. The mean embedding of a distribution $\mathbb{P}$ using a kernel k is defined as $\mu_{\mathbb{P}} = \mathbb{E}_{X \sim \mathbb{P}} k(\cdot, X)$. The mean embedding is well-defined if $\mathbb{E}_{X \sim \mathbb{P}} \sqrt{k(X, X)} < \infty$, and the mapping $\mathbb{P} \mapsto \mu_{\mathbb{P}}$ is one-to-one when k is a characteristic kernel (e.g., a universal kernel on a compact metric space $\mathcal{X}$; see Sriperumbudur et al., 2008). Therefore, for such a one-to-one mapping one can test for the equality of two distributions $\mathbb{P}_s$ and $\mathbb{P}_t$ by comparing the distance between their mean embeddings. This distance is called the maximum mean discrepancy (MMD). Given two samples x_s and x_t, the corresponding (biased) V-statistic is given by

$$\widehat{\mathrm{MMD}}_b^2 = \|\hat{\mu}_s - \hat{\mu}_t\|_{\mathcal{H}}^2 ,$$

where $\hat{\mu}_s \coloneqq n_s^{-1} \sum_{i=1}^{n_s} k(\cdot, x_{s,i})$ and similarly for $\hat{\mu}_t$. When expanded, this is exactly (5.1) with a general kernel.

There are numerous benefits to the kernel viewpoint. For example, this approach can be applied to generic sets $\mathcal{X}$ so long as a kernel can be defined on $\mathcal{X} \times \mathcal{X}$. Moreover, any kernel parameters can be fixed to obtain a consistent test, and the convergence rate does not depend on the dimension d (Gretton et al., 2012).

5.3.2 From MMD to change-point estimation

Given the MMD test statistic, there are four natural ways in which one could perform change-point estimation, each of which will be described in this section. For simplicity we will assume that the number of points in each point cloud is constant, i.e., $n_t = n$ for some n.

Binary segmentation with MMD. First, one could consider using the greedy binary segmentation approach, as done by e.g., Srivastava and Worsley (1986). This approach would

first locate the time t_1 at which the MMD statistic computed on the sets $\{x_1, \ldots, x_{t_1-1}\}$ and $\{x_{t_1}, \ldots, x_T\}$ is maximized. If the null hypothesis is rejected (i.e., the distributions of the observations within each set are different), then t_1 is identified as a change point. The procedure then performs this partitioning strategy separately on the set of point clouds before t_1 and from t_1 onward, and so on, until no more tests reject the null hypothesis. Defining the cost of evaluating the kernel k_x on a pair of inputs to be $O(c_x)$ for some c_x, the computational complexity of this approach is $O(c_x n^2 T^2)$. The downside to this approach is that it is a heuristic.

Sliding windows with MMD. To avoid the heuristic nature of binary segmentation, one could instead consider a sliding window method. Specifically, defining a window size w and overlap p, the sliding window method performs hypothesis tests using the MMD on all pairs of segments $\{x_t, \ldots, x_{t+w}\}$ and $\{x_{t+w-p}, \ldots, x_{t+2w-p}\}$ for $t = 1, 1+w-p, 1+2w-2p, \ldots, T$. The computational complexity of this approach is $O(c_x n^2 T)$. The downside to the sliding window approach is that the power of the tests is smaller than if one knew the true segment lengths.

Maximum sum of MMDs across adjacent segments. In an attempt to increase the power of the tests, one could consider setting the endpoints so as to maximize the sum of the test statistics. Defining $x_{t_j:t_{j+1}-1} = \{x_{t_j}, \ldots, x_{t_{j+1}-1}\}$, this could then entail solving

$$\max_{t_1,\ldots,t_m} \sum_{j=0}^{m-1} \frac{(t_{j+1}-t_j)(t_{j+2}-t_{j+1})}{\sum_{j'=0}^{m-1}(t_{j'+1}-t_{j'})(t_{j'+2}-t_{j'+1})} MMD_b^2(x_{t_j:t_{j+1}-1}, x_{t_{j+1}:t_{j+2}-1}) \ .$$

The problem can be solved using dynamic programming (Fisher, 1958; Bellman, 1961; Kay, 1993). Its computational complexity is $O(c_x n^2 T^2)$. To our knowledge, there has been no theoretical analysis of this approach.

Maximum sum of MMDs across all segments. Finally, consider the case where all segments are believed to have different distributions, i.e., $\mathbb{P}_s \neq \mathbb{P}_t$ for all point clouds x_s, x_t not in the same segment. In this case one could consider maximizing a weighted sum of the MMD test statistics across all pairs of segments:

$$\max_{t_1,\ldots,t_m} \sum_{j,j'=0}^{m} \frac{(t_{j+1}-t_j)(t_{j'+1}-t_{j'})}{\sum_{j=0}^{m}\sum_{j'\neq j}(t_{j+1}-t_j)(t_{j'+1}-t_{j'})} MMD_b^2(x_{t_j:t_{j+1}-1}, x_{t_{j'}:t_{j'+1}-1}) \ .$$

The computational complexity of this approach is also $O(c_x n^2 T^2)$. The following proposition shows that this is equivalent to a version of the kernel change-point method of Harchaoui and Cappé (2007).

Proposition 20. *Define* $\hat{\mu}_t \coloneqq n^{-1}\sum_{i=1}^{n}\phi_x(x_{t,i})$. *Define the kernel* $k_{\hat{\mu}} : \mathcal{H}_x \times \mathcal{H}_x \to \mathbb{R}$ *as* $k_{\hat{\mu}}(\hat{\mu}_s, \hat{\mu}_t) = n^{-2}\sum_{i,i'=1}^{n} k_x(x_{s,i}, x_{t,i'})$. *Denote its corresponding canonical feature map by* $\phi_{\hat{\mu}}$ *and its reproducing kernel Hilbert space (RKHS) by* $\mathcal{H}_{\hat{\mu}}$. *Then the argmax of the problem*

$$\max_{t_1,\ldots,t_m} \sum_{j,j'=0}^{m} \frac{(t_{j+1}-t_j)(t_{j'+1}-t_{j'})}{\sum_{j=0}^{m}\sum_{j'\neq j}(t_{j+1}-t_j)(t_{j'+1}-t_{j'})} MMD_b^2(x_{t_j:t_{j+1}-1}, x_{t_{j'}:t_{j'+1}-1})$$

is the same as the argmin of the problem

$$\min_{t_1,\ldots,t_m} \sum_{j=0}^{m}\sum_{t=t_j}^{t_{j+1}-1} \|\phi_{\hat{\mu}}(\hat{\mu}_t) - \hat{\nu}_{t_j:t_{j+1}-1}\|_{\mathcal{H}_{\hat{\mu}}}^2 \ , \tag{5.2}$$

where $\hat{\nu}_{t_j:t_{j+1}-1} \coloneqq (t_{j+1}-t_j)^{-1}\sum_{t=t_j}^{t_{j+1}-1}\phi_{\hat{\mu}}(\hat{\mu}_t)$.

Proof. The argmax of the problem

$$\max_{t_1,\ldots,t_m} \sum_{j,j'=0}^{m} \frac{(t_{j+1}-t_j)(t_{j'+1}-t_{j'})}{\sum_{j=0}^{m}\sum_{j'\neq j}(t_{j+1}-t_j)(t_{j'+1}-t_{j'})} MMD_b^2(x_{t_j:t_{j+1}-1}, x_{t_{j'}:t_{j'+1}-1})$$

is the same as the argmin of the problem

$$\min_{t_1,\ldots,t_m} -\frac{1}{2}\sum_{j,j'=0}^{m}(t_{j+1}-t_j)(t_{j'+1}-t_{j'})MMD_b^2(x_{t_j:t_{j+1}-1}, x_{t_{j'}:t_{j'+1}-1}) \; .$$

Now observe that we can rewrite this objective as

$$\begin{aligned}
&-\frac{1}{2}\sum_{j,j'=0}^{m}(t_{j+1}-t_j)(t_{j'+1}-t_{j'})MMD_b^2(x_{t_j:t_{j+1}-1}, x_{t_{j'}:t_{j'+1}-1}) \\
=& -\sum_{j=0}^{m}\frac{(t_{j+1}-t_j)\sum_{j'\neq j}(t_{j'+1}-t_{j'})}{(t_{j+1}-t_j)^2}\sum_{s,t=t_j}^{t_{j+1}-1}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'}) \\
&+\sum_{j=0}^{m}\sum_{j'\neq j}\frac{(t_{j+1}-t_j)(t_{j'+1}-t_{j'})}{(t_{j+1}-t_j)(t_{j'+1}-t_{j'})}\sum_{s=t_j}^{t_{j+1}-1}\sum_{t=t_{j'}}^{t_{j'+1}-1}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'}) \\
=& \sum_{j=0}^{m}\frac{(t_{j+1}-t_j)-T}{t_{j+1}-t_j}\sum_{s,t=t_j}^{t_{j+1}-1}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'})+\sum_{j=0}^{m}\sum_{j'\neq j}\sum_{s=t_j}^{t_{j+1}-1}\sum_{t=t_{j'}}^{t_{j'+1}-1}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'}) \\
=& \sum_{s,t=1}^{T}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'})-\sum_{j=0}^{m}\frac{T}{t_{j+1}-t_j}\sum_{s,t=t_j}^{t_{j+1}-1}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'}) \\
=& T\sum_{j=0}^{m}\left\{\sum_{t=t_j}^{t_{j+1}-1}\sum_{i,i'=1}^{n}k_x(x_{t,i},x_{t,i'})-\frac{1}{t_{j+1}-t_j}\sum_{s,t=t_j}^{t_{j+1}-1}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'})\right\} \\
&-T\left\{\sum_{t=1}^{T}\sum_{i,i'=1}^{n}k_x(x_{t,i},x_{t,i'})-\frac{1}{T}\sum_{s,t=1}^{T}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'})\right\} \; .
\end{aligned}$$

Recalling that $k_{\hat{\mu}}(\hat{\mu}_s,\hat{\mu}_t)=n^{-2}\sum_{i,i'=1}^{n}k_x(x_{s,i},x_{t,i'})$ and dividing by n^2T, the objective becomes

$$\sum_{j=0}^{m}\sum_{t=t_j}^{t_{j+1}-1}\|\phi_{\hat{\mu}}(\hat{\mu}_t)-\hat{\nu}_j\|_{\mathcal{H}_{\hat{\mu}}}^2-\sum_{t=1}^{T}\|\phi(\hat{\mu}_t)-\hat{\nu}\|_{\mathcal{H}_{\hat{\mu}}}^2 \; ,$$

where $\hat{\nu}_j=(t_{j+1}-t_j)^{-1}\sum_{t=t_j}^{t_{j+1}-1}\phi_{\hat{\mu}}(\hat{\mu}_t)$ and $\hat{\nu}=T^{-1}\sum_{t=1}^{T}\phi_{\hat{\mu}}(\hat{\mu}_t)$. As the last term is a

constant, the minimizer of this objective is the same as that of the objective

$$\sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \|\phi_{\hat{\mu}}(\hat{\mu}_t) - \hat{\nu}_j\|_{\mathcal{H}_{\hat{\mu}}}^2 \ .$$

□

In the remainder of this chapter we will consider this kernel change-point approach.

5.3.3 Kernel change-point analysis

For a fixed number of change points, the problem (5.2) minimizes the sum of the intra-segment scatters and is a generalization of the normal change-in-mean method (Kay, 1993). Now let $k_{\hat{\mu}} : \mathcal{H}_x \times \mathcal{H}_x \to \mathbb{R}$ be a generic kernel on the averages of feature maps $\hat{\mu}_t \coloneqq {n_t}^{-1} \sum_{i=1}^{n_t} \phi_x(x_{t,i}) \in \mathcal{H}_x$, $t = 1, \ldots, T$ from point clouds. Denote the canonical feature map of $k_{\hat{\mu}}$ by $\phi_{\hat{\mu}}$ and its reproducing kernel Hilbert space (RKHS) by $\mathcal{H}_{\hat{\mu}}$. In addition, let $\text{pen} : \mathbb{R}^m \to \mathbb{R}$ be a penalty on the number of change points and $m_{\max}$ be the maximum allowable number of change points. Then the penalized version of the kernel change-point method solves

$$\min_{m=1,\ldots,m_{\max}} \min_{t_1,\ldots,t_m} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}-1} \|\phi_{\hat{\mu}}(\hat{\mu}_t) - \hat{\nu}_{t_j:t_{j+1}-1}\|_{\mathcal{H}_{\hat{\mu}}}^2 + \text{pen}(m) \ , \tag{5.3}$$

where $\hat{\nu}_{t_j:t_{j+1}-1} = (t_{j+1} - t_j)^{-1} \sum_{t=t_j}^{t_{j+1}-1} \phi_{\hat{\mu}}(\hat{\mu}_t)$ (Arlot et al., 2019). The objective function can be evaluated using the kernel trick, which allows one to avoid having to explicitly work with $\phi_{\hat{\mu}}$. Assuming the cost of evaluating the kernel $k_{\hat{\mu}}$ on a pair of inputs is $O(c_{\hat{\mu}})$ for some $c_{\hat{\mu}}$, the problem can be solved in $O((c_{\hat{\mu}} + m_{\max})T^2)$ time via dynamic programming. We will now discuss choices of the kernels k_x and $k_{\hat{\mu}}$, in addition to the penalty function pen.

Kernel selection. With this approach, two kernels need to be specified: the kernel k_x on the points within each point cloud and the kernel $k_{\hat{\mu}}$ between point clouds. The kernel k_x

should generally be a characteristic kernel in order to be able to distinguish between empirical mean embeddings of different distributions. We will use the Gaussian Radial Basis Function (RBF) kernel in the experiments.

We consider two different options for the kernel $k_{\hat{\mu}}$. First, consider the case where $k_{\hat{\mu}}$ is the linear kernel. This is similar to the case considered in Proposition 20, where we showed that the kernel change-point algorithm amounts to minimizing a weighted sum of squared MMD statistics between all pairs of segments when the number of points in each point cloud is the same. Second, consider the case where $k_{\hat{\mu}}$ is a Gaussian RBF kernel. In this case the kernel change-point algorithm minimizes the intra-segment scatters when the kernel is the Gaussian RBF kernel on point clouds with the MMD as the distance metric:

$$k_{\hat{\mu}}^{\mathrm{MMD}}(\hat{\mu}_s, \hat{\mu}_t) = e^{-\frac{1}{2\sigma_{\hat{\mu}}^2}\|\hat{\mu}_s - \hat{\mu}_t\|_{\mathcal{H}_x}^2}, \tag{5.4}$$

where $\sigma_{\hat{\mu}} > 0$ is a hyperparameter.

The downside to using this approach is that it can be computationally expensive. Many kernels, including the Gaussian RBF kernel, do not have a finite-dimensional feature map. Therefore, the kernel trick must be used to evaluate the MMD. If the complexity of evaluating the kernel k_x on a pair of inputs is $O(c_x)$ for some c_x, computing $k_{\hat{\mu}}$ for every pair of point clouds has complexity $O(c_x(\max_t n_t)^2 T^2)$. Therefore, exactly computing $k_{\hat{\mu}}$ when k_x is the Gaussian kernel is impractical when n_t and T are each on the order of thousands. The computational complexity of the kernel change-point algorithm in this setting is $O((c_{\hat{\mu}} + c_x(\max_t n_t)^2)T^2)$.

Kernel approximation. Since the kernel k_x is often expensive to compute for every pair of points across every point cloud, we can instead approximate it by projecting onto a subspace (Williams and Seeger, 2000). Let $w_1, \ldots, w_p \in \mathbb{R}^d$ and consider projecting $\phi_x(x_{t,i})$ onto $\mathrm{Span}\{\phi_x(w_1), \ldots, \phi_x(w_p)\}$. This gives the problem

$$\min_{\alpha \in \mathbb{R}^p} \left\| \phi_x(x_{t,i}) - \sum_{j=1}^{p} \alpha_j \phi_x(w_j) \right\|_{\mathcal{H}_x}^2. \tag{5.5}$$

The solution to this problem is given by

$$\alpha^\star = \left([k_x(w_j, w_\ell)]_{j,\ell=1}^p\right)^{-1} [k_x(w_j, x)]_{j=1}^p \ .$$

Denoting the first term by $k_x(W, W)$ and the second by $k_x(W, x)$, we have that

$$\begin{aligned}\langle \phi(x_{t,i}), \phi(x_{t',i'}) \rangle_{\mathcal{H}_x} &\approx \left\langle \sum_{j=1}^p \alpha_j^\star \phi_x(w_j), \sum_{j=1}^p \alpha_j^{\star'} \phi_x(w_j) \right\rangle_{\mathbb{R}^p} \\ &= \alpha^{\star T} k_x(W, W) \alpha^{\star'} \\ &= \langle k_x(W, W)^{-1/2} k_x(W, x_{t,i}), k_x(W, W)^{-1/2} k_x(W, x_{t',i'}) \rangle \ ,\end{aligned}$$

where $\alpha^{\star'}$ is the solution to (5.5) when $x_{t,i}$ is replaced by $x_{t',i'}$. Hence, we may use $\psi(x_{t,i}) \coloneqq k_x(W, W)^{-1/2} k_x(W, x_{t,i})$ in place of $\phi(x_{t,i})$ for all t, i in order to approximate the kernel. The vectors $w_1, \ldots, w_p$ can be chosen by sampling from the inputs. By approximating k_x in this manner, the computational complexity of the kernel change-point algorithm is reduced to $O((c_{\hat{\mu}} + c_x p \max_t n_t + p^2 \max_t n_t) T^2 + c_x p^2 + p^3)$.

Penalty selection. The remaining component to specify is the penalty pen. There are numerous possibilities for the penalty. First, one can use a penalty on the number of change points. Arlot et al. (2019) propose a penalty of the form $\text{pen}(m) = T^{-1}[c_1 \log \binom{T-1}{m} + c_2(m+1)]$. The resultant mean estimates were proven to satisfy a non-asymptotic oracle inequality. Garreau and Arlot (2018) considered a simplified penalty, of the form $c_1 M^2 (m+1)/T$, where M is an almost-sure upper bound on $\sqrt{k_x(x, x)}$ for all $x \in \mathcal{X}$. In the setting where the frequency with which points are observed increases, the authors showed that the change-point estimates converge toward the true change points at the rate $O(\log(T)/T)$. A common way of estimating the unknown penalty parameters is via the slope heuristic (Baudry et al., 2012).

Alternatively, if other related time series exist and are annotated, then these can be used to estimate the number of change points. If there is a time series that is expected to have the same number of changes as in the time series of interest, then one can use a penalty of the

form $\text{pen}(m) = \delta_{\hat{m}}(m)$, where $\hat{m}$ is the number of change points from the other time series and $\delta_{\hat{m}}(m) = 0$ if $m = \hat{m}$ and ∞ otherwise. On the other hand, if there are related labeled time series, then one can estimate the parameters of a penalty on them and then use these parameters when estimating the number of change points in the time series of interest.

5.4 Experiments

We apply the kernel change-point method developed in Section 5.3.3 to biological data collected by an automated flow cytometer during oceanographic cruises. We aim to answer the following questions:

1. Does the method successfully estimate changes in the distribution of phytoplankton communities?

2. How sensitive is the method to the parameter settings?

3. Do the estimated change points in the biological data coincide with change points in the physical environment?

4. Can we leverage additional data on the physical environment from other cruises to estimate the number of change points?

5.4.1 Experimental details

Here we describe the data and the parameter settings of the kernel change-point method. Additional experimental details may be found in Appendix D.1.

Data. The data used in this study consists of two components. First, there is physical data, which consists of measurements of sea surface temperature and salinity collected at 3-minute intervals throughout the cruises from the ship's underway thermosalinograph system. Second, there is biological data, which consists of measurements of light scatter and fluorescence

emissions of individual particles. The biological data is organized into files recorded every 3 minutes, where each file contains measurements of the cytometric characteristics of between 1,000 and 100,000 particles ranging from 0.5 to 5 microns in diameter. The volume of data in any given file depends on the total biomass of phytoplankton within the sampled region. Each particle is characterized by two measures of fluorescence emission (chlorophyll and phycoerythrin), its diameter (estimated from light scatter measurements by the application of Mie theory for spherical particles), and its label (identified based on a combination of manual gating and a semi-supervised clustering method)(Ribalet et al., 2019). Note that we use the particle labels only for verification of our approach.

In the experiments we use standardized physical data from 18 cruises and biological data from 9 cruises (see Table D.1 in Appendix D.1). Change points in the physical data were manually annotated in order to have ground truth labels. Details regarding the manual annotation may be found in Appendix D.1.2. The physical data is available online at `https://github.com/seaflow-uw/seaflow-sfl`.

Parameter settings. The change-point estimation method discussed in Section 5.3.3 has several parameters that must be set. We set the bandwidth of the Gaussian RBF kernel k_x to the median pairwise distance between inputs, a common rule of thumb. We set the vectors used perform the projections, $w_1, \ldots, w_p$, to the centroids obtained by clustering the input data into p clusters using 100 iterations of k-means. Moreover we added a regularization term of $0.001\,\mathrm{I}_{p\times p}$ to $k_x(W, W)$ prior to the inversion. Unless otherwise specified, we fix the dimension p of the projection to 128 and use the Gaussian RBF kernel with the rule-of-thumb bandwidth for $k_{\hat{\mu}}$. In order to target phytoplankton community shifts associated with larger-scale oceanographic features, such as mesoscale eddies ($\sim$100km) and gyre boundaries, and to avoid generating a large number of change points associated with high frequency variability, we set the minimum distance between change points to be 5 samples, which represents 15 minutes or roughly 5km for a ship moving at 10 knots. For the sensitivity analysis, we allow each of these parameters to vary while fixing the others.

The notion of what the appropriate number of change points is is subjective and context-dependent; one could be interested in large changes in the phytoplankton distribution, e.g., if the fraction of one species abruptly changes by more than 20%, or one could be interested in smaller changes in distribution. Therefore, in the first set of results obtained from data collected during the KOK1606 cruise, we fix the number of change points to 10. Afterward, we estimate the number of change points based on manual annotations of change points in the physical data.

Code. The code for this chapter was written in Python and Cython (Behnel et al., 2011). The change-point estimation package we developed builds upon Yael, Scikit-learn, and Faiss (Douze and Jégou, 2014; Pedregosa et al., 2011; Johnson et al., 2019) and may be found online at `https://github.com/cjones6/chapydette`. All of the experiments, including the data cleaning, feature generation, and change-point estimation components, take under five hours total to run on a machine with 128GB of memory and a 16 core, 2.80GHz processor.

5.4.2 Results

Changes in distribution. We first aim to assess whether our method successfully estimates changes in the distribution of phytoplankton along the course of a cruise. To do so, we run our change-point estimation method on the biological measurements taken on the KOK1606 cruise when fixing the number of change points to 10. The resulting estimated change points are displayed on the x-axis of Figure 5.3.

We assess the quality of the estimated change points in three ways. First, for the change point at 2175km we plot in Figure 5.2 the distribution of the cell diameter and phycoerythrin measurements 5km before and after the estimated change point. Recall that the cell labels were not available to our method; only the cell measurements were. From the plots we can see that there is a large decrease from 2170km to 2180km in the fraction of cells that have phycoerythrin larger than 20,000 or cell diameter larger than $1\mu m$. On the other hand, there appears to be an increase in the fraction of cells that have cell diameter less than $1\mu m$

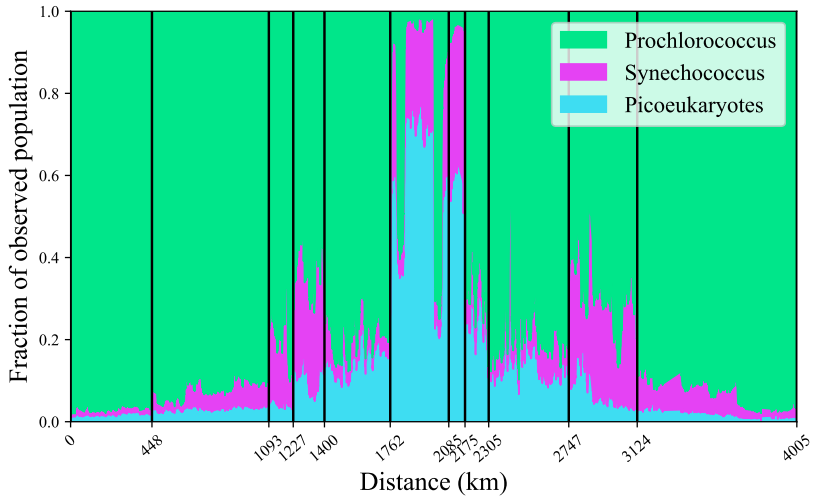


Figure 5.3: Estimated change points in the biological data from the KOK1606 cruise overlaid on the phytoplankton distribution observed during the cruise.

and phycoerythrin less than 20,000. From the labels, we can see that this corresponds to a decrease in the fraction of *Synechococcus* and picoeukaryotes and an increase in the fraction of *Prochlorococcus*.

Next, Figure 5.3 overlays the estimated change points on a plot of the phytoplankton distribution observed at each three-minute time period throughout the course of the cruise. From these plots we can see that the algorithm indeed locates large, abrupt changes in the phytoplankton community structure. For example, it is able to detect the large jump from 16% picoeukaryotes to 60% picoeukaryotes that was observed centered on the change point that our method detected from the biological data at 1762km along the cruise track.

A unique feature of the KOK1606 cruise is that it was an out-and-back cruise over a period of 3 weeks (see Figure D.1 in Appendix D.2, which displays the latitude and longitude of the research vessel as a function of the alongtrack distance). This provides us with a fortuitous method of checking the efficacy and consistency of our method, as we expect that change points detected on the way out should also be detected on the return leg. The second

change point, at 1093km, is only 35km in space from the last change point, at 3124km. The fourth change point, at 1400km, is 58km from the second-to-last change point, at 2747km. Moreover, the fifth change point, at 1762km, is 13km from the seventh change point, at 2175km. Given that the ocean is a dynamic environment subject to constant movement driven by ocean currents, it is likely that the detected change points could shift in space. Surface mean current speeds in the region of the Pacific sampled by the KOK1606 cruise are ~ 0.1 m/s (Lumpkin and Johnson, 2013), which could drive a displacement of ~ 60km over the course of a week. The close spacing between our outbound and return change points confirms that our method is able to detect the change points that were sampled more than once.

Sensitivity analysis. Next, we examine how sensitive the estimated change points are to the parameter settings. We again examine this for data from the KOK1606 cruise with 10 change points. The results are shown in Figure D.2 in Appendix D.2. Beginning with the dimension p of the projection onto the subspace in Figure D.2a, we can see that the change-point estimates are all within 8km of those obtained from the baseline $p = 128$ as long as the dimension is at least 16. In the case of $p = 4$ or 8, one estimated change point shifts to approximately 3687km, which is where another, albeit smaller, change in distribution occurred.

Next we examine the effect of the kernel $k_{\hat{\mu}}$. Comparing the Gaussian RBF kernel to the linear kernel, we see from Figure D.2b that the choice of kernel makes little difference in this setting. All of the estimated change points with the linear kernel are within 2km of those from the Gaussian RBF kernel. The choice of the Gaussian kernel bandwidth makes more of a difference. In Figure D.2c, we vary the quantity $1/(2\sigma^2)$ on a log scale, where the endpoints are determined by setting σ to the 1st and 99th percentiles of the distances between inputs. For very small bandwidths (less than 0.05), the algorithm tends to estimate change points to be where the changes in distribution are less perceptible. In addition, the algorithm finds a change point at approximately 3687km for bandwidths less than 0.2.

Finally, we examine the effect of setting a minimum distance between change points in Figure D.2d. The change points estimated with each minimum distance are the same.

Correspondence between biological and physical change points. The distribution of individual phytoplankton species is a reflection of that species' environmental niche (Hutchinson, 1957), defined as the range of environmental parameters within which a species can subsist. Niches can be controlled by physicochemical factors such as temperature and nutrient availability, as well as biotic processes such as competition and predation. In order to better understand the controls on phytoplankton distributions, we need to gain a better understanding of the balance between physical and biological controls in driving shifts in species' distributions, which are reflected as shifts in overall phytoplankton community structure. By comparing the location of physical change points (based on surface temperature and salinity) and biological change points (based on SeaFlow data), we can begin to understand how important changes in the physical environment are in controlling phytoplankton community structure. When physical and biological change points coincide, that would suggest that physical processes are driving the observed community shift, whereas biological change points that do not coincide with physical change points likely reflect community shifts driven by biotic processes.

In these experiments the biological change points are estimated as described above, and the physical change points are estimated on the standardized temperature and salinity data using the normal change-in-mean method described by Kay (1993, Chapter 12). Figure 5.4 plots the results overlaid on the temperature and salinity measurements throughout the cruise. In this plot the temperature and salinity data were smoothed using a low-pass Savitsky-Golay filter with window length 11 and order 2 (Savitzky and Golay, 1964). The estimated physical and biological change points are within 20km of each other 50% of the time. However, even when they do not quite coincide, the estimated biological change points are associated with large changes in temperature and salinity. These results suggest that shifts in phytoplankton community structure are largely associated with corresponding shifts

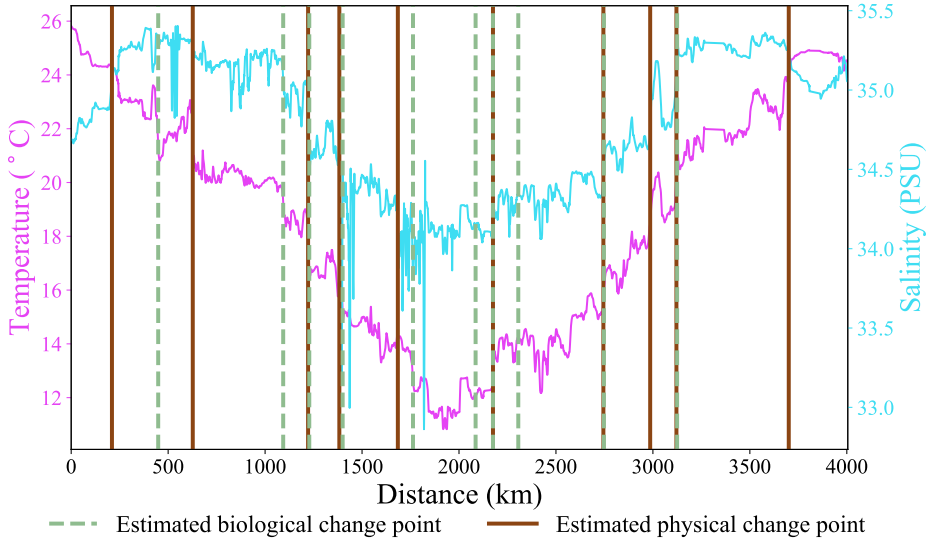


Figure 5.4: Estimated change points in the biological and physical data from the KOK1606 cruise overlaid on the temperature and salinity data recorded throughout the cruise.

in physical ocean properties. These results support previous work that showed that water masses play an important role in structuring phytoplankton communities (e.g., d'Ovidio et al., 2010).

Estimation of the number of change points. We now examine the problem of estimating the number of change points. To do so, we use the annotated physical data. For each cruise, we perform change-point estimation on the physical data with the added penalty $\alpha(m+1)/T(2\log(T/(m+1))+5)$ of Lebarbier (2005). We set the parameter α in the penalty to the value that minimizes the sum of the distances between the number of estimated and annotated change points on the physical data from all other cruises. The values considered for α are $0, 0.01, 0.02, \ldots, 1$ and the value chosen is always either 0.13 or 0.14. The results are presented in the left panel of Figure 5.5. In this case the number of estimated change points for 15 of the 18 cruises is within a factor of two of the number of annotated change points. We compare to the rule of thumb of Harchaoui and Lévy-Leduc (2007) in Figure D.3 in Appendix D.2. The rule of thumb says to choose the minimum number of change points

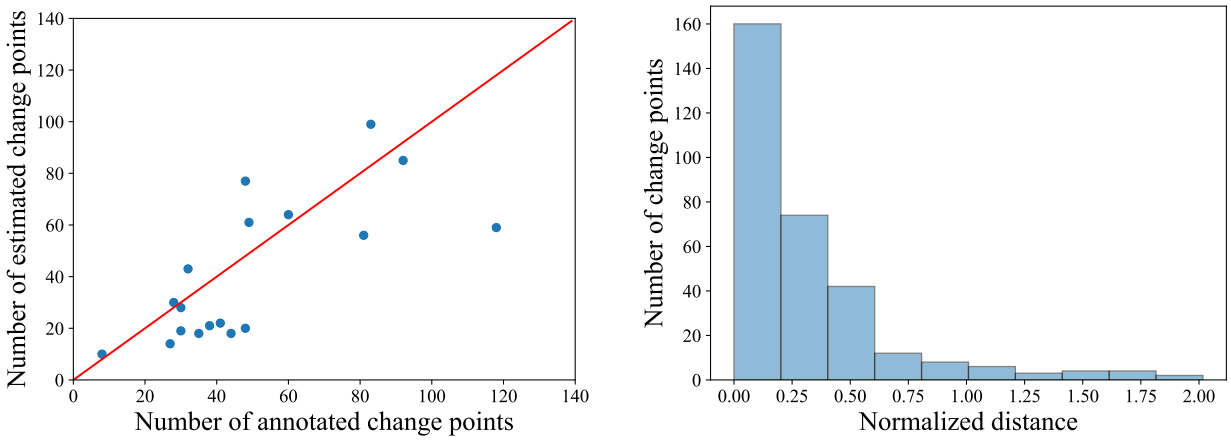


Figure 5.5: Estimated and annotated number of change points on the physical data (left) and histogram of the distances from each estimated biological change point to the nearest annotated physical change point for the same cruise, normalized by the average distance between annotated change points within the cruise (right). The diagonal red line in the plot on the left denotes the locations where the points would ideally lie.

m such that the ratio of successive objective values with $m+1$ change points and m change points exceeds $1-\nu$ for some value ν. The optimization we perform in this case is over the threshold for the ratio of successive objective values $1-\nu = 0.9, 0.901, 0.902, \ldots, 0.99$, and the value chosen is always 0.98. With this method the correlation between the number of estimated change points and the number of annotated change points is -0.02. In contrast, with the penalty-based approach the correlation is 0.73. These results suggest that the method of Lebarbier (2005) is promising in this context while the rule of thumb of Harchaoui and Lévy-Leduc (2007) is ineffective.

The right panel of Figure 5.5 plots a histogram of normalized distances from the estimated biological change points to the nearest annotated physical change point in each cruise. The distances for a cruise were normalized by the average distance between annotated change points in the given cruise in order to improve interpretability. We find that 18% of estimated change points are within 5% of the average distance between annotated change points to the nearest annotated change point. This value increases to 33% when considering annotated

change points within 10% of the average distance between change points.

Finally, Figure D.4 in Appendix D.2 displays the estimated and annotated change points for KOK1606 cruise. Using the penalty method, the estimated number of change points is 64 whereas the number of annotated change points is 60.

5.5 Conclusion

In this work we presented a kernel-based change-point estimation method that applies to point cloud data. We applied the method to data collected by a shipboard flow cytometer during research cruises. The results suggest that the method is generally insensitive to the choice of parameters and is able to locate meaningful changes in the phytoplankton distribution. Furthermore, we found that changes in the distribution of phytoplankton generally co-occur with changes in temperature and salinity. This paved the way for estimating the number of change points in the biological data based on the number of change points in the physical data. In future work it would be interesting to explore online methods for change-point detection on point cloud data that could be used in real time during research cruises to help scientists developing an adaptive sampling approach.

Chapter 6

CONCLUSION

This dissertation focused on two areas of research: deep learning and change-point estimation. The first half of the dissertation contributed to the understanding of several fundamental questions in deep learning, while the second half broadened the scope of existing change-point estimation methods.

In Chapter 2 we studied the correspondence between the layers of ConvNets and kernels. We showed how transform a ConvNet into a kernel counterpart, called a CKN, and computed the smoothness constants of CKNs. After proposing a new method for training deep networks, we compared the performance of the two types of models across four architecture/dataset combinations. We found that on three of the four architectures the CKNs perform on par with their ConvNet counterparts. One drawback of our approach is that it fails to account for the biases in ConvNets. In future work it would be interesting to compare the results to those from other kernel viewpoints, including the Gaussian process viewpoint and the neural tangent kernel viewpoint. It would also be interesting to compare the neural and kernel viewpoints on other architectures, such as recurrent or transformer networks for sequence data arising in natural language processing. More broadly, it would be interesting to leverage the kernel viewpoint to design new architectures with kernels that possess desired invariance properties.

Chapter 3 transitioned to learning feature representations for classification/clustering with any ratio of labeled to unlabeled data. In that chapter we proposed a unified framework for end-to-end learning that gracefully adapts to the level of supervision: in the unsupervised case it performs clustering, in the supervised case it performs classification, and in the semi-supervised case it performs a combination of clustering and classification. We demonstrated

on four architectures/datasets that when additional labeled data is not available but would be beneficial, using unlabeled data instead can often yield performance improvements. An open question related to this work is how to prove the convergence of the matrix balancing algorithm in the general case. Going forward, it would be interesting to apply this approach with alternative objectives as described in Section 3.3 and to think about whether something similar can be done for statistical problems with ordinal data or other data enjoying a discrete structure.

Chapter 4 began the half of the dissertation related to change-point estimation. That chapter, which built off of the ideas from Chapter 3, proposed a framework for learning feature representations for change-point estimation with any ratio of labeled to unlabeled sequences. It, too, works in the unsupervised, semi-supervised, and supervised settings. We demonstrated that learning the feature representations with this framework can result in better segmentations. However, the smoothing method we proposed for optimizing the objective showed no benefit over alternating optimization. One drawback of our approach, which we intend to address in future work, is that we assume the number of change points is known. Going forward, it would be interesting to develop a method whose complexity is sub-linear rather than quadratic in the length of the sequence. It would also be interesting to extend the method to work with non-i.i.d. data such as data with long-range dependence.

Finally, Chapter 5 proposed a kernel-based change-point estimation method for point cloud data. We applied this method to oceanographic data in an effort to better understand the structure of phytoplankton communities. One downside to this approach is that it takes into account only the distribution of the phytoplankton species, rather than the quantity of phytoplankton. Moreover, we required annotated auxiliary data in order to estimate the number of change points. In future work it would be interesting to investigate alternative methods for estimating the number of change points. It could also be interesting to adapt this method to work on other kinds of sequences of point clouds, such as sequences of images of phytoplankton. From a broader perspective, it would be ideal to develop a method that can improve based on human input during oceanographic research cruises.

BIBLIOGRAPHY

M. Abramowitz and I. A. Stegun. Handbook of mathematical functions with formulas, graphs and mathematical tables. Washington: U.S. Department of Commerce, 1964.

J.-B. Alayrac, P. Bojanowski, N. Agrawal, J. Sivic, I. Laptev, and S. Lacoste-Julien. Unsupervised learning from narrated instruction videos. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 4575–4583, 2016.

N. H. Anderson, P. Hall, and D. M. Titterington. Two-sample test statistics for measuring discrepancies between two multivariate probability density functions using kernel-based density estimates. *Journal of Multivariate Analysis*, 50(1):41–54, 1994.

D. Ardila, A. P. Kiraly, S. Bharadwaj, B. Choi, J. J. Reicher, L. Peng, D. Tse, M. Etemadi, W. Ye, G. Corrado, D. P. Naidich, and S. Shetty. End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. *Nature Medicine*, 2019.

S. Arlot, A. Celisse, and Z. Harchaoui. A kernel multiple change-point algorithm via model selection. *Journal of Machine Learning Research*, 20(162):1–56, 2019.

S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8139–8148, 2019.

S. Arora, S. S. Du, Z. Li, R. Salakhutdinov, R. Wang, and D. Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In *Proceedings of the International Conference on Learning Representations*, 2020.

Y. M. Asano, C. Rupprecht, and A. Vedaldi. Self-labelling via simultaneous clustering and representation learning. In *Proceedings of the International Conference on Learning Representations*, 2020.

J. A. D. Aston and C. Kirch. High dimensional efficiency with applications to change point tests. *Electronic Journal of Statistics*, 12(1):1901–1947, 2018.

F. R. Bach and Z. Harchaoui. DIFFRAC: a discriminative and flexible framework for clustering. In *Advances in Neural Information Processing Systems*, pages 49–56, 2007.

F. R. Bach and M. I. Jordan. Learning spectral clustering, with application to speech separation. *Journal of Machine Learning Research*, 7:1963–2001, 2006.

P. Bachman, O. Alsharif, and D. Precup. Learning with pseudo-ensembles. In *Advances in Neural Information Processing Systems*, pages 3365–3373, 2014.

P. Bachman, R. D. Hjelm, and W. Buchwalter. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pages 15509–15519, 2019.

J. Bai. Vector Autoregressive Models with Structural Changes in Regression Coefficients and in Variance-Covariance Matrices. *Annals of Economics and Finance*, 1(2):303–339, November 2000.

O. E. Barndorff-Nielsen and D. R. Cox. *Inference and asymptotics*, volume 52. London: Chapman and Hall, 1994.

M. Basseville and I. V. Nikiforov. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

J.-P. Baudry, C. Maugis, and B. Michel. Slope heuristics: overview and implementation. *Statistics and Computing*, 22(2):455–470, 2012.

L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37:1554–1563, 1966.

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.

A. Beck and M. Teboulle. Smoothing and first order methods: a unified framework. *SIAM Journal on Optimization*, 22(2):557–580, 2012.

S. Behnel, R. Bradshaw, C. Citro, L. Dalcín, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science and Engineering*, 13(2):31–39, 2011.

M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

M. Belkin, S. Ma, and S. Mandal. To understand deep learning we need to understand kernel learning. In *Proceedings of the International Conference on Machine Learning*, pages 540–548, 2018.

R. Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM*, 4:284, 1961.

D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. Raffel. MixMatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 5050–5060, 2019.

D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 3rd edition, 2016.

L. Beyer, X. Zhai, A. Oliver, and A. Kolesnikov. S4L: self-supervised semi-supervised learning. In *Proceedings of the International Conference on Computer Vision*, pages 1476–1485, 2019.

A. Bietti and J. Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations. *Journal of Machine Learning Research*, 20: 25:1–25:49, 2019.

J. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report TR-97-021, ICSI, 1997.

L. Bo and C. Sminchisescu. Efficient match kernel between sets of features for visual recognition. In *Advances in Neural Information Processing Systems*, pages 135–143, December 2009.

L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1729–1736, 2011.

R. Bock, A. Chilingarian, M. Gaug, F. Hakl, T. Hengstebeck, M. Jirina, J. Klaschka, E. Kotrc, P. Savicky, S. Towers, A. Vaicilius, and W. Wittek. Methods for multidimensional event classification: A case study using images from a Cherenkov gamma-ray telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 516(2):511–528, 2004.

P. Bojanowski and A. Joulin. Unsupervised learning by predicting noise. In *Proceedings of the International Conference on Machine Learning*, pages 517–526, 2017.

P. Bojanowski, R. Lajugie, F. Bach, I. Laptev, J. Ponce, C. Schmid, and J. Sivic. Weakly supervised action labeling in videos under ordering constraints. In *Proceedings of the European Conference on Computer Vision*, pages 628–643, 2014.

P. Bojanowski, R. Lajugie, E. Grave, F. Bach, I. Laptev, J. Ponce, and C. Schmid. Weakly-supervised alignment of video with text. In *Proceedings of the International Conference on Computer Vision*, pages 4462–4470, 2015.

N. Boumal, P.-A. Absil, and C. Cartis. Global rates of convergence for nonconvex optimization on manifolds. *IMA Journal of Numerical Analysis*, 2016.

J. V. Bouvrie, L. Rosasco, and T. A. Poggio. On invariance in hierarchical models. In *Advances in Neural Information Processing Systems*, pages 162–170, 2009.

G. Box. Robustness in the strategy of scientific model building. In *Robustness in Statistics*, pages 201 – 236. Academic Press, 1979.

B. E. Brodsky and B. S. Darkhovsky. *Nonparametric methods in change-point problems.* Dordrecht: Kluwer Academic Publishers, 1993.

J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

T. Cacoullos. Estimation of a multivariate density. *Annals of the Institute of Statistical Mathematics*, 18:179–189, 1966.

O. Cappé. Online EM algorithm for hidden Markov models. *Journal of Computational and Graphical Statistics*, 20(3):728–749, 2011.

M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision*, pages 139–156, 2018.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

W. Chang, C. Li, Y. Yang, and B. Póczos. Kernel change-point detection with auxiliary deep generative models. In *Proceedings of the International Conference on Learning Representations*, 2019.

O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning.* The MIT Press, 1st edition, 2010.

C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. Su. This looks like that: Deep learning for interpretable image recognition. In *Advances in Neural Information Processing Systems*, pages 8928–8939, 2019.

J. Chen and A. K. Gupta. *Parametric statistical change point analysis. With applications to genetics, medicine, and finance.* Boston, MA: Birkhäuser, 2nd revised and expanded edition, 2012.

L. Chizat, E. Oyallon, and F. Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems*, pages 2933–2943, 2019.

Y. Cho and L. K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, pages 342–350, 2009.

A. Christmann and I. Steinwart. Universal kernels on non-standard input spaces. In *Advances in Neural Information Processing Systems*, pages 406–414, 2010.

K. R. Clarke. Non-parametric multivariate analyses of changes in community structure. *Australian Journal of Ecology*, 18(1):117–143, 1993.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, 2009.

B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M. Balcan, and L. Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014.

A. Daniely, R. Frostig, and Y. Singer. Toward deeper understanding of neural networks: the power of initialization and a dual view on expressivity. In *Advances in Neural Information Processing Systems*, pages 2253–2261, 2016.

A. Daniely, R. Frostig, V. Gupta, and Y. Singer. Random features for compositional kernels. *CoRR*, abs/1703.07872, 2017.

D. Davis, D. Drusvyatskiy, S. Kakade, and J. D. Lee. Stochastic subgradient method converges on tame functions. *Foundations of Computational Mathematics*, Jan 2019.

S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980.

J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. ImageNet: A large-scale hierarchical image database. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

E. D. Denman and A. N. Beavers Jr. The matrix sign function and computations in systems. *Applied Mathematics and Computation*, 2:63–94, 1976.

C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the International Conference on Computer Vision*, pages 1422–1430, 2015.

A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. A. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1734–1747, 2016.

M. Douze and H. Jégou. The Yael library. In *Proceedings of the ACM international conference on Multimedia*, pages 687–690. ACM, 2014.

F. d'Ovidio, S. De Monte, S. Alvain, Y. Dandonneau, and M. Lévy. Fluid dynamical niches of phytoplankton types. *Proceedings of the National Academy of Sciences*, 107 (43):18366–18370, 2010.

G. B. Dubelaar, P. L. Gerritzen, A. E. Beeker, R. R. Jonker, and K. Tangen. Design and first results of CytoBuoy: A wireless flow cytometer for in situ analysis of marine and fresh waters. *Cytometry*, 37(4):247–254, Dec. 1999.

G. K. Dziugaite and D. M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2017.

S. Eger, P. Youssef, and I. Gurevych. Is it time to Swish? Comparing deep learning activation functions across NLP tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4415–4424, 2018.

P. G. Falkowski, R. T. Barber, and V. Smetacek. Biogeochemical controls and feedbacks on ocean primary production. *Science*, 281(5374):200–206, 1998.

P. Fearnhead. Exact and efficient Bayesian inference for multiple changepoint problems. *Statistics and Computing*, 16(2):203–213, Jun 2006.

P. Fearnhead and Z. Liu. On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):589–605, 2007.

P. Fearnhead and Z. Liu. Efficient Bayesian analysis of multiple changepoint models with dependence across segments. *Statistics and Computing*, 21(2):217–229, 2011.

P. Fearnhead, R. Maidstone, and A. Letchford. Detecting changes in slope with an L_0 penalty. *Journal of Computational and Graphical Statistics*, 28(2):265–275, 2019.

C. B. Field, M. J. Behrenfeld, J. T. Randerson, and P. Falkowski. Primary production of the biosphere: Integrating terrestrial and oceanic components. *Science*, 281(5374): 237–240, 1998.

W. D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53:789–798, 1958.

N. Flammarion, B. Palaniappan, and F. Bach. Robust discriminative clustering with sparse regularizers. *Journal of Machine Learning Research*, 18:80:1–80:50, 2017.

E. B. Fox, E. B. Sudderth, M. I. Jordan, and A. S. Willsky. A sticky HDP-HMM with application to speaker diarization. *The Annals of Applied Statistics*, 5(2A):1020–1056, 2011.

K. Fukushima. Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193, 1980.

A. B. Gardner, A. M. Krieger, G. J. Vachtsevanos, and B. Litt. One-class novelty detection for seizure analysis from intracranial EEG. *Journal of Machine Learning Research*, 7:1025–1044, 2006.

D. Garreau and S. Arlot. Consistent change-point detection with kernels. *Electronic Journal of Statistics*, 12(2):4440–4486, 2018.

A. Garriga-Alonso, L. Aitchison, and C. E. Rasmussen. Deep convolutional networks as shallow Gaussian processes. In *Proceedings of the International Conference on Learning Representations*, 2019.

K. Ghasedi Dizaji, A. Herandi, C. Deng, W. Cai, and H. Huang. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the International Conference on Computer Vision*, pages 5747–5756, 2017.

S. M. Goldfeld and R. E. Quandt. A Markov model for switching regressions. *Journal of Econometrics*, 1:3–16, 1973.

I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

A. D. Gordon and H. J. B. Birks. Numerical methods in quaternary palaeoecology I. Zonation of pollen diagrams. *New Phytologist*, 71(5):961–979, 1972.

Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems*, pages 529–536, 2004.

A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

I. Guyon, S. R. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems*, pages 545–552, 2004.

L. W. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 11(9): 1074–1085, 1992.

J. D. Hamilton. Analysis of time series subject to changes in regime. *Journal of Econometrics*, 45(1-2):39–70, 1990.

Z. Harchaoui and O. Cappé. Retrospective mutiple change-point estimation with kernels. In *Proceedings of the IEEE Workshop on Statistical Signal Processing*, pages 768–772. IEEE, 2007.

Z. Harchaoui and C. Lévy-Leduc. Catching change-points with Lasso. In *Advances in Neural Information Processing Systems*, pages 617–624, 2007.

Z. Harchaoui and C. Lévy-Leduc. Multiple change-point estimation with a total variation penalty. *Journal of the American Statistical Association*, 105(492):1480–1493, 2010.

Z. Harchaoui, F. R. Bach, and E. Moulines. Kernel change-point analysis. In *Advances in Neural Information Processing Systems*, pages 609–616, 2008.

T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical learning with sparsity. The Lasso and generalizations.* CRC Press, 2015.

P. Häusser, A. Mordvintsev, and D. Cremers. Learning by association - A versatile semi-supervised training method for neural networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 626–635, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision*, pages 770–778, 2016.

N. J. Higham. Stable iterations for the matrix square root. *Numerical Algorithms*, 15 (2):227–242, 1997.

N. J. Higham. *Functions of Matrices - Theory and Computation*. SIAM, 2008.

M. Hoai and F. De la Torre. Max-margin early event detectors. *International Journal of Computer Vision*, 107(2):191–202, 2014.

R. Hosseini and S. Sra. An alternative to EM for Gaussian mixture models: batch and stochastic Riemannian optimization. *Mathematical Programming*, 181(1 (A)):187–223, 2020.

P. Huang, H. Avron, T. N. Sainath, V. Sindhwani, and B. Ramabhadran. Kernel methods match deep neural networks on TIMIT. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 205–209, 2014.

D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

G. E. Hutchinson. Concluding remarks. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 22, 1957.

J. Hyrkas, S. Clayton, F. Ribalet, D. Halperin, E. V. Armbrust, and B. Howe. Scalable clustering algorithms for continuous environmental flow cytometry. *Bioinformatics*, 32 (3):417–423, 2015.

A. Hyvärinen and H. Morioka. Unsupervised feature extraction by time-contrastive learning and nonlinear ICA. In *Advances in Neural Information Processing Systems*, pages 3765–3773, 2016.

A. Iscen, G. Tolias, Y. Avrithis, and O. Chum. Label propagation for deep semi-supervised learning. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 5070–5079, 2019.

A. Jacot, C. Hongler, and F. Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, pages 8580–8589, 2018.

A. Jalali, Q. Han, I. Dumitriu, and M. Fazel. Exploiting tradeoffs for exact recovery in heterogeneous stochastic block models. In *Advances in Neural Information Processing Systems*, pages 4871–4879, 2016.

Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio. Fantastic generalization measures and where to find them. In *Proceedings of the International Conference on Learning Representations*, 2020.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data (early access)*, 2019.

C. Jones and Z. Harchaoui. End-to-end learning for retrospective change-point estimation. In *IEEE International Workshop on Machine Learning for Signal Processing (to appear)*, 2020.

A. Joulin and F. R. Bach. A convex relaxation for weakly supervised classifiers. In *Proceedings of the International Conference on Machine Learning*, 2012.

A. Joulin, F. R. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1943–1950, 2010.

B. Juang and L. R. Rabiner. The segmental k-means algorithm for estimating parameters of hidden Markov models. *Proceedings of the IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(9):1639–1641, 1990.

K. Kamnitsas, D. C. Castro, L. L. Folgoc, I. Walker, R. Tanno, D. Rueckert, B. Glocker, A. Criminisi, and A. V. Nori. Semi-supervised learning via compact latent space clustering. In *Proceedings of the International Conference on Machine Learning*, pages 2464–2473, 2018.

R. M. Karp. Reducibility among combinatorial problems. *Kiberneticheskiui Sbornik. Novaya Seriya*, 12:16–38, 1975.

S. M. Kay. *Fundamentals of statistical signal processing.* Prentice Hall PTR, 1993.

H. K. Khalil. *Nonlinear systems.* New York, NY: Macmillan Publishing Company, 1992.

A. Y. Kim, C. Marzban, D. B. Percival, and W. Stuetzle. Using labeled data to evaluate change detectors in a multivariate streaming environment. *Signal Processing*, 89(12): 2529 – 2536, 2009.

G. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions and stochastic processes. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

S. I. M. Ko, T. T. L. Chong, and P. Ghosh. Dirichlet process hidden Markov multiple change-point model. *Bayesian Analysis*, 10(2):275–296, 2015.

P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks. In *Proceedings of the International Conference on Learning Representations*, 2016.

A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1106–1114, 2012.

T. L. Lai and H. Xing. Sequential change-point detection when the pre- and post-change parameters are unknown. *Sequential Analysis*, 29(2):162–175, 2010.

R. Lajugie, F. R. Bach, and S. Arlot. Large-margin metric learning for constrained partitioning problems. In *Proceedings of the International Conference on Machine Learning*, pages 297–305, 2014.

E. Lebarbier. Detecting multiple change-points in the mean of Gaussian process by model selection. *Signal Processing*, 85(4):717–736, 2005.

Y. LeCun. *Modeles connexionnistes de l'apprentissage*. PhD thesis, Université P. et M. Curie (Paris 6), June 1987.

Y. LeCun. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, 1988.

Y. LeCun. Generalization and network design strategies. In *Connectionism in perspective*. Elsevier, 1989.

Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404, 1989.

Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 53–60, 1995.

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and M. K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

D.-H. Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *International Conference on Machine Learning Workshop on Challenges in Representation Learning*, 2013.

J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as Gaussian processes. In *Proceedings of the International Conference on Learning Representations*, 2018.

J. Lee, L. Xiao, S. S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, pages 8570–8581, 2019.

S. Li, Y. Xie, H. Dai, and L. Song. Scan B-statistic for kernel change-point detection. *Sequential Analysis*, 38(4):503–544, 2019.

Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox. DeepIM: Deep iterative matching for 6D pose estimation. In *European Conference on Computer Vision*, pages 695–711, 2018.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3 (B)):503–528, 1989.

H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.

D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, pages 1150–1157, 1999.

S. Löwe, P. O'Connor, and B. Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, pages 3033–3045, 2019.

R. Lumpkin and G. C. Johnson. Global ocean surface velocities from drifters: Mean, variance, el niño–southern oscillation response, and seasonal cycle. *Journal of Geophysical Research: Oceans*, 118(6):2992–3006, 2013.

H. Lütkepohl. *Handbook of matrices.* Chichester: John Wiley & Sons, 1996.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

J. Mairal. End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pages 1399–1407, 2016.

J. Mairal, F. R. Bach, and J. Ponce. Sparse modeling for image and vision processing. *Foundations and Trends in Computer Graphics and Vision*, 8(2-3):85–283, 2014a.

J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *Advances in Neural Information Processing Systems*, pages 2627–2635, 2014b.

C. D. Manning and H. Schütze. *Foundations of statistical natural language processing.* MIT Press, 2001.

D. S. Matteson and N. A. James. A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, 109(505): 334–345, 2014.

A. Matthews, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani. Gaussian process behaviour in wide deep neural networks. In *Proceedings of the International Conference on Learning Representations*, 2018.

M. Meila and J. Shi. A random walks view of spectral segmentation. In *Proceedings of the Workshop on Artificial Intelligence and Statistics*, 2001.

M. Meila, S. M. Shortreed, and L. Xu. Regularized spectral learning. In *Proceedings of the Workshop on Artificial Intelligence and Statistics*, 2005.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning.* Adaptive computation and machine learning. MIT Press, 2012.

J. J. Moreau. Proximité et dualité dans un espace hilbertien. *Bulletin de la Société Mathématique de France*, 93:273–299, 1965.

M. Nagano, T. Nakamura, T. Nagai, D. Mochihashi, I. Kobayashi, and W. Takano. High-dimensional motion segmentation by variational autoencoder and Gaussian processes. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 105–111, 2019.

R. M. Neal. *Bayesian Learning for Neural Networks.* New York, NY: Springer, 1996.

Y. Nesterov. *Introductory lectures on convex optimization. A basic course.* Boston: Kluwer Academic Publishers, 2004.

Y. Nesterov. *Lectures on convex optimization.* Springer, 2nd edition, 2018.

M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *Proceedings of the European Conference on Computer Vision*, pages 69–84, 2016.

R. Novak, L. Xiao, Y. Bahri, J. Lee, G. Yang, D. A. Abolafia, J. Pennington, and J. Sohl-dickstein. Bayesian deep convolutional networks with many channels are Gaussian processes. In *Proceedings of the International Conference on Learning Representations*, 2019.

D. Oglic and T. Gärtner. Nyström method with kernel k-means++ samples as landmarks. In *Proceedings of the International Conference on Machine Learning*, pages 2652–2660, 2017.

S. M. Oh, J. M. Rehg, T. R. Balch, and F. Dellaert. Learning and inferring motion patterns using parametric segmental switching linear dynamic systems. *Proceedings of the International Journal of Computer Vision*, 77(1-3):103–124, 2008.

A. Oliver, A. Odena, C. A. Raffel, E. D. Cubuk, and I. J. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *Advances in Neural Information Processing Systems*, pages 3239–3250, 2018.

R. J. Olson, A. Shalapyonok, and H. M. Sosik. An automated submersible flow cytometer for analyzing pico- and nanophytoplankton: FlowCytobot. *Deep Sea Research Part I: Oceanographic Research Papers*, 50(2):301–315, Feb. 2003.

E. Oyallon, E. Belilovsky, and S. Zagoruyko. Scaling the scattering transform: Deep hybrid networks. In *Proceedings of the International Conference on Computer Vision*, pages 5619–5628, 2017.

E. Oyallon, S. Zagoruyko, G. Huang, N. Komodakis, S. Lacoste-Julien, M. B. Blaschko, and E. Belilovsky. Scattering networks for hybrid representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2208–2221, 2019.

E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035. 2019.

M. Paulin, J. Mairal, M. Douze, Z. Harchaoui, F. Perronnin, and C. Schmid. Convolutional patch representations for image retrieval: An unsupervised approach. *Proceedings of the International Journal of Computer Vision*, 121(1):149–168, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

K. Pelckmans, J. De Brabanter, J. Suykens, and B. De Moor. Convex clustering shrinkage. In *Proceedings of the PASCAL Workshop on Statistics and Optimization of Clustering*, 2005.

F. Perez-Cruz and O. Bousquet. Kernel methods and their potential use in signal processing. *IEEE Signal Processing Magazine*, 21(3):57–65, May 2004.

M. E. Peters, W. Ammar, C. Bhagavatula, and R. Power. Semi-supervised sequence tagging with bidirectional language models. In *Association for Computational Linguistics*, pages 1756–1765, 2017.

G. Peyré and M. Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

V. K. Pillutla, V. Roulet, S. M. Kakade, and Z. Harchaoui. A smoother way to train structured prediction models. *CoRR*, abs/1902.03228, 2019.

L. Plaugic. FaceApp's creator apologizes for the app's skin-lightening 'hot' filter. *The Verge*, 2017.

D. Potapov, M. Douze, Z. Harchaoui, and C. Schmid. Category-specific video summarization. In *Proceedings of the European Conference on Computer Vision*, pages 540–555, 2014.

R. E. Quandt. The estimation of the parameters of a linear regression system obeying two separate regimes. *Journal of the American Statistical Association*, 53:873–880, 1958.

R. E. Quandt. Tests of the hypothesis that a linear regression system obeys two separate regimes. *Journal of the American Statistical Association*, 55:324–330, 1960.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184, 2007.

P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. In *International Conference on Learning Representations (Workshop Track)*, 2018.

C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449, 2017.

F. Ribalet, C. Berthiaume, A. Hynes, J. Swalwell, M. Carlson, S. Clayton, G. Hennon, C. Poirier, E. Shimabukuro, A. White, and E. V. Armbrust. Seaflow data v1, high-resolution abundance, size and biomass of small phytoplankton in the North Pacific. *Scientific Data*, 6(1):277, Nov 2019.

J. Risley. Microsoft's millennial chatbot Tay.ai pulled offline after Internet teaches her racism. *GeekWire*, 2016.

R. T. Rockafellar. Convex analysis. Princeton University Press, 1970.

M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27:832–837, 1956.

V. Roulet. *Sur la géométrie de problèmes d'optimisation et leur structure. (On the geometry of optimization problems and their structure)*. PhD thesis, École Normale Supérieure, Paris, France, 2017.

A. Rudi and L. Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems*, pages 3215–3225, 2017.

Y. Saatcci, R. D. Turner, and C. E. Rasmussen. Gaussian process change point models. In *Proceedings of the International Conference on Machine Learning*, pages 927–934, 2010.

M. Sangnier, J. Gauthier, and A. Rakotomamonjy. Early and reliable event detection using proximity space representation. In *Proceedings of the International Conference on Machine Learning*, pages 2310–2319, 2016.

A. Savitzky and M. J. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

S. Saxena and J. Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, pages 4053–4061, 2016.

M. Scetbon and Z. Harchaoui. Harmonic decompositions of convolutional networks. In *Proceedings of the International Conference on Machine Learning (to appear)*, 2020.

J. Schmidt-Hieber. Nonparametric regression using deep neural networks with ReLU activation function. *Annals of Statistics*, 48(4):1875–1897, 08 2020.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning series. MIT Press, 2002.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, and S. Levine. Time-contrastive networks: Self-supervised learning from video. In *Proceedings of the International Conference on Robotics and Automation*, pages 1134–1141, 2018.

V. Shankar, A. Fang, W. Guo, S. Fridovich-Keil, L. Schmidt, J. Ragan-Kelley, and B. Recht. Neural kernels without tangents. In *Proceedings of the International Conference on Machine Learning (to appear)*, 2020.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*, 2015.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *International Conference on Learning Representations (Workshop Track)*, 2014.

R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21:343–348, 1967.

L. Song, J. Huang, A. J. Smola, and K. Fukumizu. Hilbert space embeddings of conditional distributions with applications to dynamical systems. In *Proceedings of the International Conference on Machine Learning*, pages 961–968, 2009.

R. Song, M. Banerjee, and M. R. Kosorok. Asymptotics for change-point models under varying degrees of mis-specification. *The Annals of Statistics*, 44(1):153–182, 2016.

H. M. Sosik, R. J. Olson, and E. V. Armbrust. Flow Cytometry in Phytoplankton Research. In *Chlorophyll a Fluorescence in Aquatic Sciences: Methods and Applications*, pages 171–185. Springer Netherlands, 2010.

G. W. Soules. The rate of convergence of Sinkhorn balancing. *Linear algebra and its applications*, 150:3–40, 1991.

V. Spokoiny. Multiscale local change point detection with applications to value-at-risk. *Annals of Statistics*, 37(3):1405–1436, 2009.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *International Conferenceon Learning Representations (Workshop Track)*, 2015.

B. K. Sriperumbudur, A. Gretton, K. Fukumizu, G. R. G. Lanckriet, and B. Schölkopf. Injective Hilbert space embeddings of probability measures. In *Proceedings of the Conference on Learning Theory*, pages 111–122, 2008.

M. S. Srivastava and K. J. Worsley. Likelihood ratio tests for a change in the multivariate normal mean. *Journal of the American Statistical Association*, 81(393):199–204, 1986.

I. Steinwart and A. Christmann. *Support vector machines*. New York, NY: Springer, 2008.

J. E. Swalwell, F. Ribalet, and E. V. Armbrust. Seaflow: A novel underway flow-cytometer for continuous observations of phytoplankton in the ocean. *Limnology and Oceanography: Methods*, 9(10):466–477, 2011.

C. Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 526–527, 2004.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations*, 2014.

R. Szeliski. *Computer Vision - Algorithms and Applications*. Texts in Computer Science. Springer, 2011.

A. Tartakovsky, I. Nikiforov, and M. Basseville. *Sequential analysis: Hypothesis testing and changepoint detection*, volume 136. CRC Press, 2015.

A. G. Tartakovsky, B. L. Rozovskii, R. B. Blazek, and H. Kim. A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. *IEEE Transactions on Signal Processing*, 54(9):3372–3382, 2006.

J. Thickstun, Z. Harchaoui, D. P. Foster, and S. M. Kakade. Invariances and data augmentation for supervised music transcription. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2241–2245, 2018.

C. Truong, L. Oudre, and N. Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167, 2020.

A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018.

L. Van Der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson,

..., and Y. Vázquez-Baeza. Scipy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, Mar 2020.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

K. von Frisch. *The Dance Language and Orientation of Bees*. Harvard University Press, 1993.

X. Wang and A. Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the International Conference on Computer Vision*, pages 2794–2802, 2015.

H. White. Maximum likelihood estimation of misspecified models. *Econometrica*, 50(1): 1–25, 1982.

M. White and D. Schuurmans. Generalized optimal reverse prediction. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 1305–1313, 2012.

C. K. I. Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems*, pages 295–301, 1996.

C. K. I. Williams and M. W. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, pages 682–688, 2000.

J. Williams. Algorithm 232 (Heapsort). *Communications of the ACM*, 7(6):347–348, 1964.

L. Wu, D. Wang, and Q. Liu. Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems*, pages 10655–10665, 2019.

Z. Wu and R. M. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

Z. Wu, Y. Xiong, S. X. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

J. Xie, R. B. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the International Conference on Machine Learning*, pages 478–487, 2016.

S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.

E. P. Xing and M. I. Jordan. On semidefinite relaxation for normalized k-cut and connections to spectral clustering. Technical Report UCB/CSD-03-1265, EECS Department, University of California, Berkeley, 2003.

L. Xu, M. White, and D. Schuurmans. Optimal reverse prediction: a unified perspective on supervised, unsupervised and semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*, pages 1137–1144, 2009.

G. Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *CoRR*, abs/1902.04760, 2019.

J. Yang, D. Parikh, and D. Batra. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 5147–5156, 2016.

Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. J. Smola, L. Song, and Z. Wang. Deep fried convnets. In *Proceedings of the International Conference on Computer Vision*, pages 1476–1483, 2015.

W. Zaremba, A. Gretton, and M. B. Blaschko. B-test: A non-parametric, low variance kernel two-sample test. In *Advances in Neural Information Processing Systems*, pages 755–763, 2013.

R. Zass and A. Shashua. Doubly stochastic normalization for spectral clustering. In *Advances in Neural Information Processing Systems*, pages 1569–1576, 2006.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pages 818–833, 2014.

H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon. Spectral relaxation for k-means clustering. In *Advances in Neural Information Processing Systems*, pages 1057–1064, 2001.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations*, 2017.

R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *Proceedings of the European Conference on Computer Vision*, pages 649–666, 2016.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2017.

C. Zou, G. Yin, L. Feng, and Z. Wang. Nonparametric maximum likelihood approach to multiple change-point problems. *The Annals of Statistics*, 42(3):970–1002, 2014.

Appendix A

APPENDIX FOR CHAPTER 2

A.1 Mathematical Description of ConvNets

A convolutional neural network architecture is typically described by its main components, including convolutions, nonlinearities, and pooling. In the existing literature, some components might be described in detail, while some others might be only roughly described. Sometimes some critical components are omitted. There might be many reasons for this situation (space limitations in particular), and the details can often be filled in by experienced practitioners. Yet, we believe that the lack of a systematic and detailed description of landmark convolutional network architectures has been an obstacle to their mathematical and statistical understanding and to the development of competing methods based on different approaches.

For a high-level review of ConvNets and their components, the reader may consult Rawat and Wang (2017). In this section, we provide what we believe to be the first mathematical descriptions of several landmark architectures in the history of convolutional neural networks. We describe each component through mathematical formulas and provide the values of all quantities involved. We used these detailed descriptions to reproduce each architecture in our experimental comparisons. Unless otherwise specified, the weight matrices or filters W_i and biases b_i discussed below are learned via gradient-based supervised training using gradient back-propagation. In contrast to Section 2.4 and Appendix A.4, here and in Appendix A.3 we describe the features at each layer of the ConvNets and CKNs using tensors for clarity of exposition.

A.1.1 LeNets

We begin with LeNet-1 and LeNet-5 (LeCun et al., 1995, 2001), which were arguably the first modern versions of ConvNets. These networks used convolutional layers and pooling/subsampling layers. LeNet-1 and LeNet-5 differ from typical modern ConvNets because the pooling is average pooling with learnable weights, the pooling is followed by the application of a nonlinearity, and the second convolutional layer uses an incomplete connection scheme. The description of LeNet-5 here differs slightly from the one given in the original paper, as the original paper used an RBF layer as the last layer rather than a fully connected layer, as one would have in more recent architectures.

The activation functions used throughout the LeNet architectures are scaled tanh functions of the form $f(x) = 1.7159 \tanh(2/3x)$. This scaling is such that $f(1) \approx 1$ and $f(-1) \approx -1$. It was argued (LeCun, 1989; LeCun et al., 1998, 2001) that these choices speed up convergence for several reasons. First, for standardized inputs f will output values with variance approximately equal to one. In addition, the second derivatives of f are largest in absolute value at ± 1. Assuming the target outputs are ± 1 this implies that near the optimum when the predictions are close to ± 1 the gradients change faster. It was noted by LeCun (1989) that this parameterization is for convenience and does not necessarily improve performance.

LeNet-1

We begin by detailing LeNet-1. Let $F_0 \in \mathbb{R}^{1\times 28\times 28}$ be the initial representation of the image. Let $W_1 \in \mathbb{R}^{4\times 1\times 5\times 5}$ and $b_1 \in \mathbb{R}^4$. The first convolutional layer convolves F_0 and W_1, adds a bias term, and then applies a pointwise nonlinearity, resulting in $F_1 \in \mathbb{R}^{4\times 24\times 24}$:

$$F_1(c,\cdot,\cdot) = 1.7159 \tanh\left(\frac{2}{3}\left[F_0(1,\cdot,\cdot) \star W_1(c,1,\cdot,\cdot) + b_1(c)\right]\right)$$

for $c = 1,\ldots,4$ where $\star$ denotes the convolution operation and tanh is understood to be applied element-wise.

Next, the second layer consists of average pooling with learnable parameters and subsam-

pling, followed by the application of a nonlinearity. Let $w_2, b_2 \in \mathbb{R}^4$ and let $e_i \in \mathbb{R}^{23}$ be a vector with entry 1 in element i and 0 elsewhere. Define $\mathcal{E}_2 = (e_1, e_3, \ldots, e_{23})$ and $\mathbb{1}_{d\times d}$ to be a matrix of ones of size $d \times d$. Then the second layer computes $F_2 \in \mathbb{R}^{4\times 12\times 12}$ given by

$$F_2(c,\cdot,\cdot) = 1.7159 \tanh\left(\frac{2}{3}\mathcal{E}_2^T\left[\left(F_1(c,\cdot,\cdot) \star \frac{1}{4}\mathbb{1}_{2\times 2}\right) w_2(c) + b_2(c)\mathbb{1}_{23\times 23}\right]\mathcal{E}_2\right),$$

for $c = 1, \ldots, 4$.

The third layer is a convolutional layer with an incomplete connection scheme between the filters at the previous layer and at this layer. Define the matrix

$$C_3 = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

to be the connection scheme. Moreover, suppose $W_3 \in \mathbb{R}^{12\times 4\times 5\times 5}$ and $b_3 \in \mathbb{R}^{12}$. The output of layer 3 is then given by $F_3 \in \mathbb{R}^{12\times 8\times 8}$, with

$$F_3(c,\cdot,\cdot) = 1.7159 \tanh\left(\frac{2}{3}\sum_{z=1}^{4}\left\{[F_2(z,\cdot,\cdot) \star W_3(c,z,\cdot,\cdot) + b_3(c)\mathbb{1}_{8\times 8}]\right\} C_3(c,z)\right)$$

for $c = 1, \ldots, 12$.

The fourth layer is analogous to the previous subsampling layer. Let $w_4, b_4 \in \mathbb{R}^{12}$ and $\mathcal{E}_4 = (e_1, e_3, e_5, e_7)$. Here $e_i \in \mathbb{R}^7$ with a 1 in element i and 0 elsewhere for all $i = 1, \ldots, 7$. Then we obtain $F_4 \in \mathbb{R}^{12\times 4\times 4}$ with

$$F_4(c,\cdot,\cdot) = 1.7159 \tanh\left(\frac{2}{3}\mathcal{E}_4^T\left[\left(F_3(c,\cdot,\cdot) \star \frac{1}{4}\mathbb{1}_{2\times 2}\right) w_4(c) + b_4(c)\mathbb{1}_{7\times 7}\right]\mathcal{E}_4\right),$$

for $c = 1, \ldots, 12$ where tanh is again understood to be applied element-wise.

Finally, the last layer is a fully-connected layer. Let $W_5 \in \mathbb{R}^{10\times 12\times 4\times 4}$ and $b_5 \in \mathbb{R}^{10}$. Then

the output is given by $F_5 \in \mathbb{R}^{10}$ with

$$F_5(c) = \sum_{z=1}^{12} [F_4(z, \cdot, \cdot) \star W_5(c, z, \cdot, \cdot) + b_5(c)]$$

for $c = 1, \ldots, 10$.

LeNet-5

Now we describe LeNet-5. Let $F_0 \in \mathbb{R}^{1\times32\times32}$ be the initial representation of the image. Let $W_1 \in \mathbb{R}^{6\times1\times5\times5}$ and $b_1 \in \mathbb{R}^6$. The first convolutional layer convolves F_0 and W_1, adds a bias term, and then applies a pointwise nonlinearity, resulting in $F_1 \in \mathbb{R}^{6\times28\times28}$:

$$F_1(c, \cdot, \cdot) = 1.7159 \tanh\left(\frac{2}{3}[F_0(1, \cdot, \cdot) \star W_1(c, 1, \cdot, \cdot) + b_1(c)]\right)$$

for $c = 1, \ldots, 6$, where tanh is understood to be applied element-wise.

Next, the second layer consists of average pooling with learnable parameters and subsampling, followed by the application of a nonlinearity. Let $w_2, b_2 \in \mathbb{R}^6$ and let $e_i \in \mathbb{R}^{27}$ be a vector with entry 1 in element i and 0 elsewhere. Define $\mathcal{E}_2 = (e_1, e_3, \ldots, e_{27})$ and $\mathbb{1}_{d\times d}$ to be a matrix of ones of size $d \times d$. Then the second layer computes $F_2 \in \mathbb{R}^{6\times14\times14}$ given by

$$F_2(c, \cdot, \cdot) = 1.7159 \tanh\left(\frac{2}{3}\mathcal{E}_2^T\left[\left(F_1(c, \cdot, \cdot) \star \frac{1}{4}\mathbb{1}_{2\times2}\right) w_2(c) + b_2(c)\mathbb{1}_{27\times27}\right]\mathcal{E}_2\right),$$

for $c = 1, \ldots, 6$.

The third layer is a convolutional layer with an incomplete connection scheme between

the filters at the previous layer and at this layer. Define the matrix

$$C_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}^T$$

to be the connection scheme. Moreover, suppose $W_3 \in \mathbb{R}^{16\times6\times5\times5}$ and $b_3 \in \mathbb{R}^{16}$. The output of layer 3 is then given by $F_3 \in \mathbb{R}^{16\times10\times10}$, with

$$F_3(c,\cdot,\cdot) = 1.7159 \tanh\left(\frac{2}{3}\left[\sum_{z=1}^{6}\left\{[F_2(z,\cdot,\cdot) \star W_3(c,z,\cdot,\cdot) + b_3(c)\mathbb{1}_{10\times10}]\right\} C_3(c,z)\right]\right).$$

for $c = 1,\ldots,16$.

The fourth layer is analogous to the previous subsampling layer. Let $w_4, b_4 \in \mathbb{R}^{16}$ and $\mathcal{E}_4 = (e_1, e_3, \ldots, e_9)$. Here $e_i \in \mathbb{R}^9$ with a 1 in element i and 0 elsewhere for all $i = 1,\ldots,9$. Then we obtain $F_4 \in \mathbb{R}^{16\times5\times5}$ with

$$F_4(c,\cdot,\cdot) = 1.7159 \tanh\left(\frac{2}{3}\mathcal{E}_4^T\left[\left(F_3(c,\cdot,\cdot) \star \frac{1}{4}\mathbb{1}_{2\times2}\right) w_4(c) + b_4(c)\mathbb{1}_{9\times9}\right]\mathcal{E}_4\right),$$

for $c = 1,\ldots,16$.

The fifth layer is a fully connected layer. Let $W_5 \in \mathbb{R}^{120\times16\times5\times5}$ and $b_5 \in \mathbb{R}^{120}$. Then the output is given by $F_5 \in \mathbb{R}^{120}$ with

$$F_5(c) = 1.7159 \tanh\left(\frac{2}{3}\left(\sum_{z=1}^{16}[F_4(z,\cdot,\cdot) \star W_5(c,z,\cdot,\cdot) + b_5(c)]\right)\right)$$

for $c = 1,\ldots,120$.

The sixth layer is also a fully connected layer. Let $W_6 \in \mathbb{R}^{84\times120}$ and $b_6 \in \mathbb{R}^{84}$. Then the

output is given by $F_6 \in \mathbb{R}^{84}$ with

$$F_6 = 1.7159 \tanh\left(\frac{2}{3}\left(W_6 F_5 + b_6\right)\right) .$$

Finally, the output layer is also a fully connected layer. Let $W_7 \in \mathbb{R}^{10\times 84}$ and $b_7 \in \mathbb{R}^{10}$. Then the output is given by $F_7 \in \mathbb{R}^{10}$ with

$$F_7 = W_7 F_6 + b_7 .$$

A.1.2 All-CNN-C

Springenberg et al. (2015) were the first to make the claim that pooling is unnecessary. Below we describe the architecture of the All-CNN-C model they presented, which consists of convolutions and ReLUs.

The input to the model is an image $F_0 \in \mathbb{R}^{3\times 32\times 32}$. Let $W_1 \in \mathbb{R}^{96\times 3\times 3\times 3}$ and $b_1 \in \mathbb{R}^{96}$ and define $Z_1 \in \mathbb{R}^{32\times 34}$ such that $(Z_1)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 32$ and is 0 otherwise. The first layer zero pads F_0 and then convolves the result with W_1 with a 1×1 stride. It then adds a bias term and applies a ReLU activation, resulting in $F_1 \in \mathbb{R}^{96\times 32\times 32}$ given by

$$F_1(c, \cdot, \cdot) = \max\left(\sum_{z=1}^{3}\left[(Z_1^T F_0(z, \cdot, \cdot) Z_1) \star W_1(c, z, \cdot, \cdot) + b_1(c)\mathbb{1}_{32\times 32}\right], 0\right),$$

for $c = 1, \ldots, 96$. Here the max is understood to be applied element-wise.

The second layer is of the same form as the first layer. Let $W_2 \in \mathbb{R}^{96\times 96\times 3\times 3}$ and $b_2 \in \mathbb{R}^{96}$ and define $Z_2 \in \mathbb{R}^{32\times 34}$ such that $(Z_2)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 32$ and is 0 otherwise. The second layer outputs $F_2 \in \mathbb{R}^{96\times 32\times 32}$ given by

$$F_2(c, \cdot, \cdot) = \max\left(\sum_{z=1}^{96}\left[(Z_2^T F_1(z, \cdot, \cdot) Z_2) \star W_2(c, z, \cdot, \cdot) + b_2(c)\mathbb{1}_{32\times 32}\right], 0\right),$$

for $c = 1, \ldots, 96$.

The third layer is a convolutional layer with a 2×2 stride. This layer acts as a replacement for a max pooling layer. Let $W_3 \in \mathbb{R}^{96\times96\times3\times3}$ and $b_3 \in \mathbb{R}^{96}$ and define $\mathcal{E}_3 = (e_1, e_3, e_5, \ldots, e_{29})$ where $e_i \in \mathbb{R}^{30}$ is a vector with 1 in element i and 0 elsewhere. The third layer outputs $F_3 \in \mathbb{R}^{96\times15\times15}$ given by

$$F_3(c,\cdot,\cdot) = \max\left(\mathcal{E}_3^T\left\{\sum_{z=1}^{96}\left[F_2(z,\cdot,\cdot)\star W_3(c,z,\cdot,\cdot) + b_3(c)\mathbb{1}_{30\times30}\right]\right\}\mathcal{E}_3, 0\right),$$

for $c = 1, \ldots, 96$.

The fourth layer returns to being a convolutional layer with a 1×1 stride, but has 192 filters. Let $W_4 \in \mathbb{R}^{192\times96\times3\times3}$ and $b_4 \in \mathbb{R}^{192}$ and define $Z_4 \in \mathbb{R}^{15\times17}$ such that $(Z_4)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 15$ and is 0 otherwise. The fourth layer outputs $F_4 \in \mathbb{R}^{192\times15\times15}$ given by

$$F_4(c,\cdot,\cdot) = \max\left(\sum_{z=1}^{96}\left[(Z_4^T F_3(z,\cdot,\cdot)Z_4)\star W_4(c,z,\cdot,\cdot) + b_4(c)\mathbb{1}_{15\times15}\right], 0\right),$$

for $c = 1, \ldots, 192$.

The fifth layer is similar to the fourth layer. Let $W_5 \in \mathbb{R}^{192\times192\times3\times3}$ and $b_5 \in \mathbb{R}^{192}$ and define $Z_5 \in \mathbb{R}^{15\times17}$ such that $(Z_5)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 15$ and is 0 otherwise. The fifth layer outputs $F_5 \in \mathbb{R}^{192\times15\times15}$ given by

$$F_5(c,\cdot,\cdot) = \max\left(\sum_{z=1}^{192}\left[(Z_5^T F_4(z,\cdot,\cdot)Z_5)\star W_5(c,z,\cdot,\cdot) + b_5(c)\mathbb{1}_{15\times15}\right], 0\right),$$

for $c = 1, \ldots, 192$.

The sixth layer is similar to the third layer, except it has 192 filters. Let $W_6 \in \mathbb{R}^{192\times192\times3\times3}$ and $b_6 \in \mathbb{R}^{192}$ and define $\mathcal{E}_6 = (e_1, e_3, e_5, \ldots, e_{13})$ where $e_i \in \mathbb{R}^{13}$ is a vector with 1 in element i and 0 elsewhere. The sixth layer outputs $F_6 \in \mathbb{R}^{192\times7\times7}$ given by

$$F_6(c,\cdot,\cdot) = \max\left(\mathcal{E}_6^T\left\{\sum_{z=1}^{192}\left[F_5(z,\cdot,\cdot)\star W_6(c,z,\cdot,\cdot) + b_6(c)\mathbb{1}_{13\times13}\right]\right\}\mathcal{E}_6, 0\right),$$

for $c = 1, \ldots, 192$.

The seventh layer is once again like the fifth layer. Let $W_7 \in \mathbb{R}^{192\times192\times3\times3}$ and $b_7 \in \mathbb{R}^{192}$ and define $Z_7 \in \mathbb{R}^{7\times9}$ such that $(Z_7)_{ij} = 1$ if $j = i+1$ for $i = 1, \ldots, 7$ and is 0 otherwise. The seventh layer outputs $F_7 \in \mathbb{R}^{192\times7\times7}$ given by

$$F_7(c,\cdot,\cdot) = \max\left(\sum_{z=1}^{192}\left[(Z_7^T F_6(z,\cdot,\cdot)Z_7)\star W_7(c,z,\cdot,\cdot) + b_7(c)\mathbb{1}_{7\times7}\right], 0\right),$$

for $c = 1, \ldots, 192$.

The eighth layer has 1×1 convolutions rather than 3×3 convolutions. Let $W_8 \in \mathbb{R}^{192\times192}$ and $b_8 \in \mathbb{R}^{192}$. The eighth layer outputs $F_8 \in \mathbb{R}^{192\times7\times7}$ given by

$$F_8(c,\cdot,\cdot) = \max\left(\sum_{z=1}^{192}\left[F_7(z,\cdot,\cdot)\star W_8(c,z) + b_8(c)\mathbb{1}_{7\times7}\right], 0\right),$$

for $c = 1, \ldots, 192$.

The ninth layer again has 1×1 convolutions but has only ten filters. Let $W_9 \in \mathbb{R}^{10\times192}$ and $b_9 \in \mathbb{R}^{10}$. The ninth layer outputs $F_9 \in \mathbb{R}^{10\times7\times7}$ given by

$$F_9(c,\cdot,\cdot) = \max\left(\sum_{z=1}^{192}\left[F_8(z,\cdot,\cdot)\star W_9(c,z) + b_9(c)\mathbb{1}_{7\times7}\right], 0\right),$$

for $c = 1, \ldots, 10$.

Next, the tenth layer is a global average pooling layer. In this layer, all of the pixels in a given feature map are averaged. The result is given by

$$F_{10}(c) = F_9(c,\cdot,\cdot)\star\frac{1}{49}\mathbb{1}_{7\times7}.$$

In the original work, a softmax function is applied to the last layer. In this work, however, we will include a fully connected layer between the tenth layer and the softmax function so that the analogous CKN will have trainable unconstrained parameters. Defining $W_{11} \in \mathbb{R}^{10\times10}$

and $b_{11} \in \mathbb{R}^{10}$, the output of our eleventh layer is thus

$$F_{11} = W_{11}F_{10} + b_{11}.$$

A.1.3 AlexNet

The AlexNet model revolutionized the field of deep learning by drastically outperforming the other competitors in the ImageNet 2012 competition. Several versions of AlexNet exist (Krizhevsky et al., 2012; Krizhevsky, 2014). Here we describe the architecture from Krizhevsky (2014) as implemented in the online code.[1] The architecture consists of convolutions, ReLU nonlinearities, pooling, and local response normalization.

The input to the model is an image $F_0 \in \mathbb{R}^{3\times224\times224}$. Let $W_1 \in \mathbb{R}^{64\times3\times11\times11}$ and $b_1 \in \mathbb{R}^{64}$. Define the zero-padding matrix $Z_1 \in \mathbb{R}^{224\times228}$ such that $(Z_1)_{ij} = 1$ if $j = i+2$ for $i = 1, \ldots, 224$ and is 0 otherwise. Moreover, define $\mathcal{E}_1 = (e_1, e_5, e_9, \ldots, e_{217})$ where $e_i \in \mathbb{R}^{218}$ is a vector with 1 in element i and 0 elsewhere. The first layer initially zero pads F_0 and then convolves F_0 and W_1 using a 4×4 stride. It then adds a bias term, downsamples, and applies a ReLU nonlinearity. This results in

$$F_1'(c,\cdot,\cdot) = \max\left\{\sum_{z=1}^{3} \mathcal{E}_1^T \left[(Z_1^T F_0(z,\cdot,\cdot)Z_1) \star W_1(c,z,\cdot,\cdot) + b_1(c)\mathbb{1}_{218\times218}\right]\mathcal{E}_1, 0\right\}$$

for $c = 1, \ldots, 64$, where the max is understood to be applied element-wise. Next, local response normalization across feature maps with a neighborhood of size 5 is performed. This results in

$$F_1''(c,i,j) = \frac{F_1'(c,i,j)}{\left(\alpha_1 + \beta_1 \sum_{c'=\max\{1,c-2\}}^{\min\{64,c+2\}} F_1'(c',i,j)^2\right)^{\gamma_1}},$$

for all c, i, j where α_1, β_1, and γ_1 are hyperparameters that can be learned. Finally, pooling is performed with a size of 3×3 and stride 2, resulting in $F_1 \in \mathbb{R}^{64\times27\times27}$. Define $\mathcal{E}_{1p} =$

[1] See `https://github.com/akrizhevsky/cuda-convnet2/blob/master/layers/layers-imagenet-1gpu.cfg`. The number of filters at the fourth convolutional layer is 256 in the code versus 384 reported in the paper.

$(e_1, e_3, e_5, \ldots, e_{53})$ where $e_i \in \mathbb{R}^{53}$ is a vector with 1 in element i and 0 elsewhere. The pooling results in F_1 given by

$$F_1(c, \cdot, \cdot) = \mathcal{E}_{1p}^T \left[\max_{i',j':i' \in \{i-1,i,i+1\}, j' \in \{j-1,j,j+1\}} F_1''(c, i', j') \right]_{i,j=2}^{54} \mathcal{E}_{1p}$$

for $c = 1, \ldots, 64$.

The second layer is similar to the first layer, with a convolution, followed by the application of a ReLU, local response normalization, and pooling. Let $W_2 \in \mathbb{R}^{192 \times 64 \times 5 \times 5}$ and $b_2 \in \mathbb{R}^{192}$. Define the zero-padding matrix $Z_2 \in \mathbb{R}^{27 \times 31}$ such that $(Z_2)_{ij} = 1$ if $j = i + 2$ for $i = 1, \ldots, 27$ and is 0 otherwise. The second layer initially zero pads F_1 and then convolves F_1 and W_2 with a 1×1 stride. It then adds a bias term and applies the ReLU nonlinearity. This results in

$$F_2'(c, \cdot, \cdot) = \max \left\{ \sum_{z=1}^{64} \left[(Z_2^T F_1(z, \cdot, \cdot) Z_2) \star W_2(c, z, \cdot, \cdot) + b_2(c) \mathbb{1}_{27 \times 27} \right], 0 \right\}$$

for $c = 1, \ldots, 192$. Next, local response normalization across feature maps with a neighborhood of size 5 is performed. This results in

$$F_2''(c, i, j) = \frac{F_2'(c, i, j)}{\left(\alpha_2 + \beta_2 \sum_{c'=\max\{1,c-2\}}^{\min\{192,c+2\}} F_2'(c', i, j)^2 \right)^{\gamma_2}},$$

for all c, i, j where α_2, β_2, and γ_2 are hyperparameters that can be learned. It then pools with a size of 3×3 and stride 2, resulting in $F_2 \in \mathbb{R}^{192 \times 13 \times 13}$. Define $\mathcal{E}_{2p} = (e_1, e_3, e_5, \ldots, e_{25})$ where $e_i \in \mathbb{R}^{25}$ is a vector with 1 in element i and 0 elsewhere. The pooling results in F_2 given by

$$F_2(c, \cdot, \cdot) = \mathcal{E}_{2p}^T \left[\max_{i',j':i' \in \{i-1,i,i+1\}, j' \in \{j-1,j,j+1\}} F_2''(c, i', j') \right]_{i,j=2}^{26} \mathcal{E}_{2p}$$

for $c = 1, \ldots, 192$.

The third layer is simpler, consisting of just a padded convolution followed by a ReLU

nonlinearity. Let $W_3 \in \mathbb{R}^{384\times192\times3\times3}$ and $b_3 \in \mathbb{R}^{384}$ and define $Z_3 \in \mathbb{R}^{13\times15}$ such that $(Z_3)_{ij} = 1$ if $j = i+1$ for $i = 1,\ldots,13$ and is 0 otherwise. The third layer outputs $F_3 \in \mathbb{R}^{384\times13\times13}$ given by

$$F_3(c,\cdot,\cdot) = \max\left(\sum_{z=1}^{192}\left[(Z_3^T F_2(z,\cdot,\cdot)Z_3)\star W_3(c,z,\cdot,\cdot) + b_3(c)\mathbb{1}_{13\times13}\right], 0\right),$$

for $c = 1,\ldots,384$.

The fourth layer is similar to the third layer. Let $W_4 \in \mathbb{R}^{256\times384\times3\times3}$ and $b_4 \in \mathbb{R}^{256}$ and define $Z_4 \in \mathbb{R}^{13\times15}$ such that $(Z_4)_{ij} = 1$ if $j = i+1$ for $i = 1,\ldots,13$ and is 0 otherwise. The fourth layer outputs $F_4 \in \mathbb{R}^{256\times13\times13}$ given by

$$F_4(c,\cdot,\cdot) = \max\left(\sum_{z=1}^{384}\left[(Z_4^T F_3(z,\cdot,\cdot)Z_4)\star W_4(c,z,\cdot,\cdot) + b_4(c)\mathbb{1}_{13\times13}\right], 0\right),$$

for $c = 1,\ldots,256$.

The fifth layer introduces max pooling again. Let $W_5 \in \mathbb{R}^{256\times256\times3\times3}$ and $b_5 \in \mathbb{R}^{256}$ and define $Z_5 \in \mathbb{R}^{13\times15}$ such that $(Z_5)_{ij} = 1$ if $j = i+1$ for $i = 1,\ldots,13$ and is 0 otherwise. The output of the convolution followed by the nonlinearity is given by

$$F_5'(c,\cdot,\cdot) = \max\left\{\sum_{z=1}^{256}\left[(Z_5^T F_4(z,\cdot,\cdot)Z_5)\star W_5(c,z,\cdot,\cdot) + b_5(c)\mathbb{1}_{13\times13}\right], 0\right\},$$

for $c = 1,\ldots,256$. Define $\mathcal{E}_{5p} = (e_1, e_3, e_5, \ldots, e_{13})$ where $e_i \in \mathbb{R}^{13}$ is a vector with 1 in element i and 0 elsewhere. The pooling results in $F_5 \in \mathbb{R}^{256\times6\times6}$ given by

$$F_5(c,\cdot,\cdot) = \mathcal{E}_{5p}^T\left[\max_{i',j':i'\in\{i-1,i,i+1\},j'\in\{j-1,j,j+1\}} F_5'(c,i',j')\right]_{i,j=2}^{12}\mathcal{E}_{5p},$$

for $c = 1,\ldots,256$.

The last three layers are fully connected layers. Let $W_6 \in \mathbb{R}^{4096\times256\times6\times6}$ and $b_6 \in \mathbb{R}^{4096}$.

Then the output of layer six is given by $F_6 \in \mathbb{R}^{4096}$ with

$$F_6(c) = \max\left(\sum_{z=1}^{256} \left[F_5(z,\cdot,\cdot) \star W_6(c,z,\cdot,\cdot) + b_6(c)\right], 0\right)$$

for $c = 1, \ldots, 4096$.

Next, let $W_7 \in \mathbb{R}^{4096\times4096}$ and $b_7 \in \mathbb{R}^{4096}$. Then the output of layer seven is given by $F_7 \in \mathbb{R}^{4096}$ with

$$F_7 = \max(W_7 F_6 + b_7, 0) \ .$$

Finally, the output layer is also a fully connected layer. Let $W_8 \in \mathbb{R}^{10\times4096}$ and $b_8 \in \mathbb{R}^{10}$. Then the output is given by $F_8 \in \mathbb{R}^{10}$ with

$$F_8 = W_8 F_7 + b_8 \ .$$

A.2 LeNet Incomplete Connection Scheme

The LeNet architectures included incomplete connection schemes at the C3 (second convolutional) layer. With these schemes, the feature maps in the C3 layers were only connected to a subset of the feature maps at the previous layer. The incomplete connection scheme was primarily motivated by computational considerations given the state-of-the-art computing resources available at the time. However, the incomplete connection scheme was also argued to break symmetry in the network owing to its highly asymmetric spatial pattern (LeCun et al., 2001).

To investigate the effect of the incomplete connection scheme, we trained the LeNet-1 and LeNet-5 models with both complete and incomplete connection schemes. The number of filters at each layer was set to be the same as in the original architectures, and the batch size was set to 16,384. The validation was performed as described in Section 2.5.1 on the models with the incomplete schemes. The resultant parameter values were then used for both the

	Incomplete scheme	Complete scheme
LeNet-1	0.9849 (0.0022)	0.9824 (0.0015)
LeNet-5	0.9885 (0.0010)	0.9890 (0.0007)

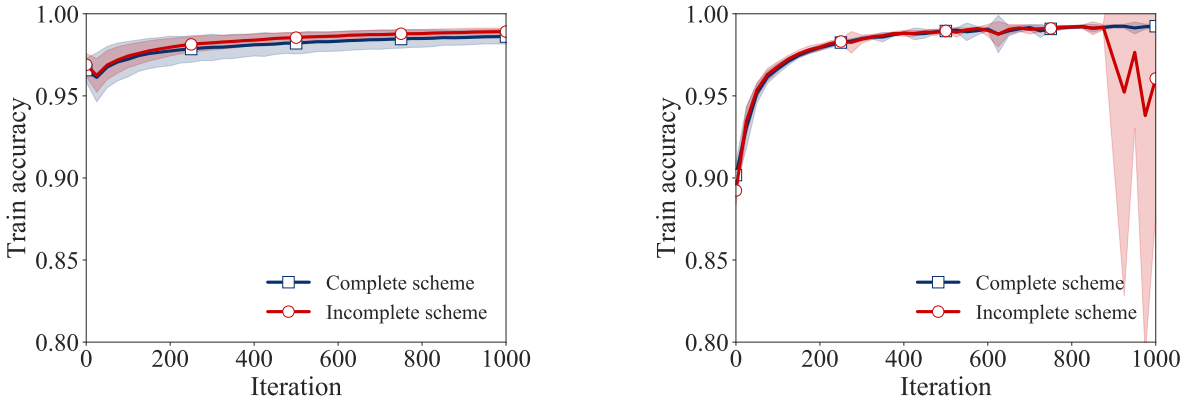
Table A.1: Average accuracy across 10 trials of training the LeNet ConvNets with incomplete and complete connection schemes at the C3 layers. The standard deviations are in parentheses.

complete and incomplete schemes.

Table A.1 reports the final accuracies on the test set of MNIST. While the incomplete connection scheme performs 0.25% better on average on LeNet-1, the two schemes are comparable on LeNet-5. Figure A.1 demonstrates that the learning generally proceeds similarly regardless of whether a complete or incomplete connection scheme is used. In the case of LeNet-5 with the incomplete connection scheme the step size appears to have become too large at around iteration 900. We occasionally observe drops in accuracy of this form, and these appear regardless of whether we use ULR-SGO or stochastic gradient optimization. Note that our LeNet-1 architecture outperforms the original LeNet-1 result (98.3%), which was on 16×16 images with padding. Our implementation of LeNet-5 with the incomplete connection scheme achieved an accuracy of 98.85%, which is slightly outside the reported range of $99.05 \pm 0.1\%$ reported by LeCun et al. (2001). This is likely due to the fact that we used 10,000 images as a validation set and did not retrain on all 60,000 images after performing the validation. As the benefit to using the incomplete connection schemes is only marginal at best, we use the models with complete connection schemes in this chapter.

A.3 CKN Counterparts to ConvNets

In this section we describe in detail the CKN counterparts to the ConvNets from Appendix A.1. Unless otherwise specified, the weight matrices or filters W_i discussed below are initially chosen in our experiments via spherical k-means and then later trained using gradient back-propagation. In these descriptions we take the kernel to be the Gaussian RBF kernel on the sphere, as this is the kernel used in the experiments. However, other dot product kernels,



(a) LeNet-1

(b) LeNet-5

Figure A.1: Average training accuracy as a function of the number of iterations across 10 trials of training the LeNet ConvNets with incomplete and complete connection schemes at the C3 layers. The error bands represent one standard deviation from the mean.

such as the arc-cosine kernel of order 1, could be used instead. For clarity of the exposition, we set the regularization parameter $\epsilon = 0$ at each layer. A pictorial representation of each architecture is given after its mathematical description.

A.3.1 LeNets

Recall from Section 2.3 that we may approximate kernels by projecting onto a subspace. The dimension of this projection corresponds to the number of filters. We use these facts to define the CKN counterparts to LeNet-1 and LeNet-5.

As mentioned in Appendix A.2, the incomplete connection schemes in the LeNets were originally motivated by computational considerations given the state-of-the-art computing resources available at the time. As the corresponding networks with complete connection schemes perform similarly, we choose to create the LeNet CKNs with complete connection schemes.

LeNet-1

Let $F_0 \in \mathbb{R}^{1\times 28\times 28}$ denote the initial representation of an image. The first layer is the counterpart to a convolutional layer and consists of applying a kernel and projecting onto a subspace. Let $W_1 \in \mathbb{R}^{4\times 1\times 5\times 5}$. Let N_1 be a matrix containing the squared norms of 5×5 patches with a 1×1 stride:

$$N_1 = [F_0 \odot F_0] \star \mathbb{1}_{5\times 5} \ ,$$

where $\odot$ is the Hadamard product. Let $k : [-1, 1] \to \mathbb{R}$ be defined as $k(z) = \exp((z-1)/\sigma^2)$. For $c = 1, \ldots, 4$, let

$$F_1'(c, \cdot, \cdot) = F_0(1, \cdot, \cdot) \star W_1(c, 1, \cdot, \cdot) \ ,$$

where $\star$ denotes the convolution operation. Then the output of the first layer is given by $F_1 \in \mathbb{R}^{4\times 24\times 24}$ with

$$F_1(\cdot, i, j) = N_1(i, j)^{1/2} k\left(\sum_{n,p=1}^{5} W_1(\cdot, 1, n, p) W_1(\cdot, 1, n, p)^T\right)^{-1/2} k\left(N_1(i, j)^{-1/2} F_1'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 24$.

Next, the second layer in the ConvNet performs average pooling and subsampling with learnable weights and then applies a pointwise nonlinearity (tanh). The corresponding CKN pools and subsamples and then applies an approximate feature map of a kernel on 1×1 patches and projects onto a subspace. Define $\mathcal{E}_2 = (e_1, e_3, e_5, \ldots, e_{23})$ where $e_i \in \mathbb{R}^{23}$ is a vector with 1 in element i and 0 elsewhere. The pooling and subsampling result in $F_2' \in \mathbb{R}^{4\times 12\times 12}$ given by

$$F_2'(c, \cdot, \cdot) = \mathcal{E}_2^T \left(F_1(c, \cdot, \cdot) \star \frac{1}{4}\mathbb{1}_{2\times 2}\right) \mathcal{E}_2$$

for $c = 1, \ldots, 4$. Next, let $W_2 \in \mathbb{R}^{4\times 4}$ be the identity matrix and define N_2 to be a matrix

containing the squared norms of 1×1 patches with a 1×1 stride

$$N_2 = \sum_{c=1}^{4} \left[F_2'(c, \cdot, \cdot) \odot F_2'(c, \cdot, \cdot) \right] .$$

Defining $F_2''(i, j, m) = F_2'(\cdot, i, j)^T W_2(m, \cdot)$ for all $i, j = 1, \ldots, 12$ and $m = 1, \ldots, 4$, the output of the second layer is then $F_2 \in \mathbb{R}^{4 \times 12 \times 12}$ given by

$$F_2(\cdot, i, j) = N_2(i, j)^{1/2} \left(\left[k \left(W_2(m, \cdot)^T W_2(n, \cdot) \right) \right]_{m,n=1}^{4} \right)^{-1/2} \left[k \left(N_2(i, j)^{-1/2} F_2''(i, j, m) \right) \right]_{m=1}^{4}$$

for $i, j = 1, \ldots, 12$.

The third layer in LeNet-1 is again a convolutional layer. As noted earlier, we use a complete connection scheme for the CKN. Therefore, this layer again consists of applying a kernel and projecting onto a subspace. Let $W_3 \in \mathbb{R}^{12 \times 4 \times 5 \times 5}$ and let N_3 be a matrix containing the squared norms of $4 \times 5 \times 5$ patches with a 1×1 stride:

$$N_3 = \sum_{c=1}^{4} \left\{ \left[F_2(c, \cdot, \cdot) \odot F_2(c, \cdot, \cdot) \right] \star \mathbb{1}_{5 \times 5} \right\} .$$

For $c = 1, \ldots, 12$, define

$$F_3'(c, \cdot, \cdot) = \sum_{z=1}^{4} F_2(z, \cdot, \cdot) \star W_3(c, z, \cdot, \cdot).$$

Then the output of the third layer is given by $F_3 \in \mathbb{R}^{12 \times 8 \times 8}$ with

$$F_3(\cdot, i, j) = N_3(i, j)^{1/2} k \left(\sum_{m=1}^{4} \sum_{n,p=1}^{5} W_3(\cdot, m, n, p) W_3(\cdot, m, n, p)^T \right)^{-1/2} k \left(N_3(i, j)^{-1/2} F_3'(\cdot, i, j) \right) .$$

The fourth layer is similar to the second layer. The CKN pools and subsamples and then applies a kernel on 1×1 patches and projects onto a subspace. Define $\mathcal{E}_4 = (e_1, e_3, e_5, e_7)$ where $e_i \in \mathbb{R}^7$ is a vector with 1 in element i and 0 elsewhere. The pooling and subsampling

result in $F_4' \in \mathbb{R}^{12\times 4\times 4}$ given by

$$F_4'(c,\cdot,\cdot) = \mathcal{E}_4^T \left(F_3(c,\cdot,\cdot) \star \frac{1}{4}\mathbb{1}_{2\times 2} \right) \mathcal{E}_4$$

for $c = 1,\ldots,4$. Next, let $W_4 \in \mathbb{R}^{12\times 12}$ be the identity matrix and define N_4 to be a matrix containing the squared norms of 1×1 patches with a 1×1 stride

$$N_4 = \sum_{c=1}^{12} \left[F_4'(c,\cdot,\cdot) \odot F_4'(c,\cdot,\cdot) \right] \ .$$

Defining $F_4''(i,j,m) = F_4'(\cdot,i,j)^T W_4(m,\cdot)$ for all $i,j = 1,\ldots,4$ and $m = 1,\ldots,12$, the output of the fourth layer is then $F_4 \in \mathbb{R}^{12\times 4\times 4}$ given by

$$F_4(\cdot,i,j) = N_4(i,j)^{1/2} \left(\left[k\left(\langle W_4(m,\cdot), W_4(n,\cdot) \rangle \right) \right]_{m,n=1}^{12} \right)^{-1/2} \left[k\left(N_4(i,j)^{-1/2} F_4''(i,j,m) \right) \right]_{m=1}^{12}$$

for $i,j = 1,\ldots,4$. The output from this layer is the set of features provided to a classifier.

LeNet-5

The CKN counterpart of LeNet-5 is similar to that of LeNet-1. Let $F_0 \in \mathbb{R}^{1\times 32\times 32}$ denote the initial representation of an image. The first layer is the counterpart to a convolutional layer and consists of applying a kernel and projecting onto a subspace. Let $W_1 \in \mathbb{R}^{6\times 1\times 5\times 5}$ and let N_1 be a matrix containing the squared norms of 5×5 patches with a 1×1 stride:

$$N_1 = \left[F_0 \odot F_0 \right] \star \mathbb{1}_{5\times 5} \ ,$$

where $\odot$ is the Hadamard product. For $c = 1,\ldots,6$, let

$$F_1'(c,\cdot,\cdot) = F_0(1,\cdot,\cdot) \star W_1(c,1,\cdot,\cdot)$$

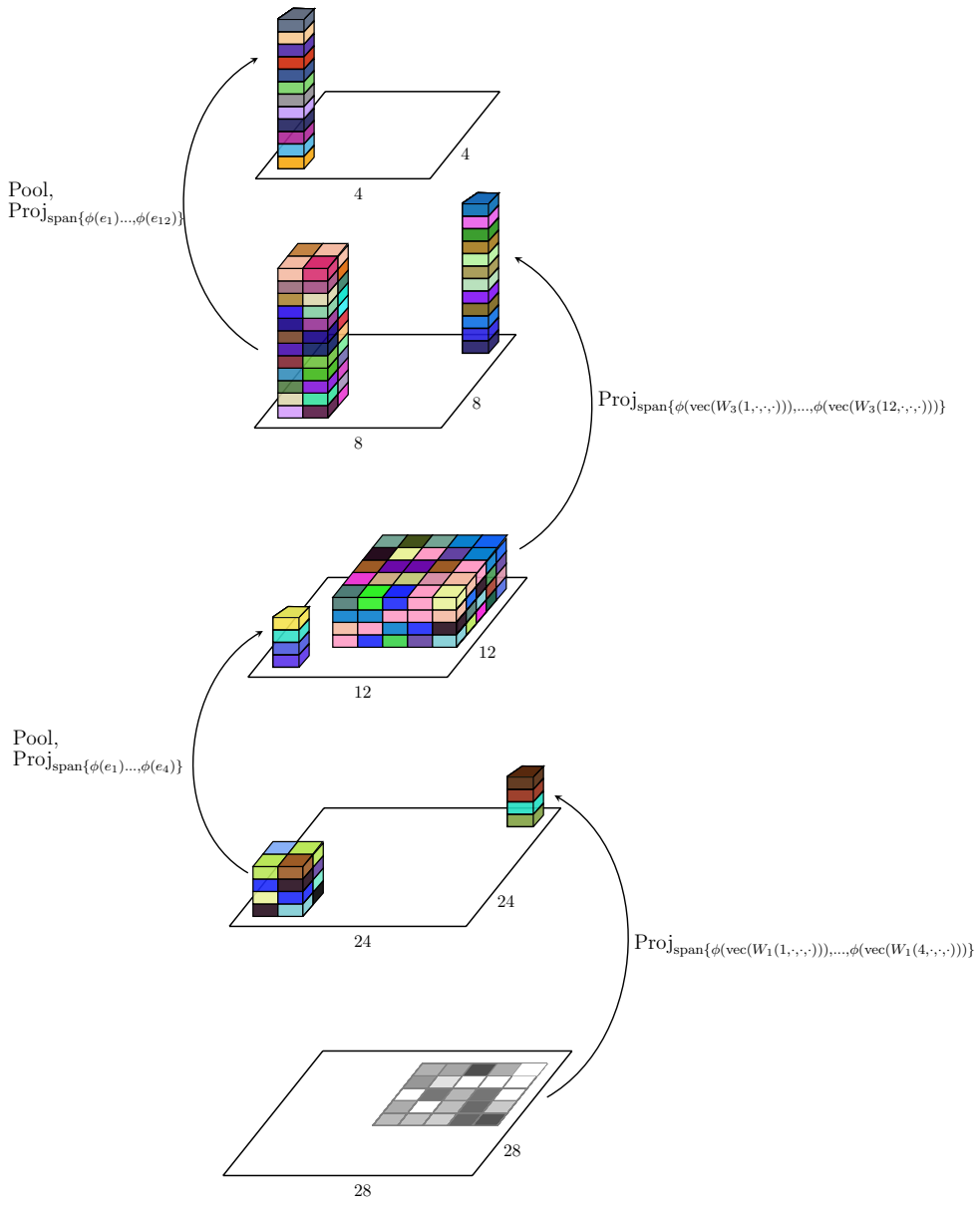


Figure A.2: LeNet-1 architecture. Each parallelogram is a 2D representation of a feature representation at a given layer. For clarity only a portion of each feature representation is displayed in 3D (in the blocks). The spatial dimensions of the stacks of blocks at the tails of the arrows are the dimensions of the filters at each layer. The height of each stack denotes the number of filters. The arrows indicate how a block gets transformed into the block at the next layer. The numbers on the sides of the parallelograms indicate the spatial dimensions of the feature representations at each layer. The colors represent the different values of the elements in the feature representations.

and define $k : [-1, 1] \to \mathbb{R}$ as $k(z) = \exp((z-1)/\sigma^2)$. Then the output of the first layer is given by $F_1 \in \mathbb{R}^{6\times 28\times 28}$ with

$$F_1(\cdot, i, j) = N_1(i, j)^{1/2} k\left(\sum_{m,n=1}^{5} W_1(\cdot, 1, m, n) W_1(\cdot, 1, m, n)^T\right)^{-1/2} k\left(N_1(i, j)^{-1/2} F_1'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 28$.

Next, the second layer in the ConvNet performs average pooling and subsampling with learnable weights and then applies a pointwise nonlinearity (tanh). The corresponding CKN pools and subsamples and then applies a kernel on 1×1 patches and projects onto a subspace. Define $\mathcal{E}_2 = (e_1, e_3, e_5, \ldots, e_{27})$ where $e_i \in \mathbb{R}^{27}$ is a vector with 1 in element i and 0 elsewhere. The pooling and subsampling result in $F_2' \in \mathbb{R}^{6\times 14\times 14}$ given by

$$F_2'(c, \cdot, \cdot) = \mathcal{E}_2^T \left(F_1(c, \cdot, \cdot) \star \frac{1}{4}\mathbb{1}_{2\times 2}\right) \mathcal{E}_2$$

for $c = 1, \ldots, 6$. Next, let $W_2 \in \mathbb{R}^{6\times 6}$ be the identity matrix and define N_2 to be a matrix containing the squared norms of 1×1 patches with a 1×1 stride

$$N_2 = \sum_{c=1}^{6} \left[F_2'(c, \cdot, \cdot) \odot F_2'(c, \cdot, \cdot)\right] .$$

Defining $F_2''(i, j, m) = F_2'(\cdot, i, j)^T W_2(m, \cdot)$ for all $i, j = 1, \ldots, 14$ and $m = 1, \ldots, 6$, the output of the second layer is then $F_2 \in \mathbb{R}^{6\times 14\times 14}$ given by

$$F_2(\cdot, i, j) = N_2(i, j)^{1/2} \left(\left[k\left(W_2(m, \cdot)^T W_2(n, \cdot)\right)\right]_{m,n=1}^{6}\right)^{-1/2} \left[k\left(N_2(i, j)^{-1/2} F_2''(i, j, m)\right)\right]_{m=1}^{6}$$

for $i, j = 1, \ldots, 14$.

The third layer in LeNet-5 is again a convolutional layer. As noted earlier, we use a complete connection scheme for the CKN. Therefore, this layer again consists of applying a kernel and projecting onto a subspace. Let $W_3 \in \mathbb{R}^{16\times 6\times 5\times 5}$ and let N_3 be a matrix containing

the squared norms of $6 \times 5 \times 5$ patches with a 1×1 stride:

$$N_3 = \sum_{c=1}^{6} \left\{ [F_2(c, \cdot, \cdot) \odot F_2(c, \cdot, \cdot)] \star \mathbb{1}_{5\times 5} \right\} .$$

For $c = 1, \ldots, 16$, let

$$F_3'(c, \cdot, \cdot) = \sum_{z=1}^{6} F_2(z, \cdot, \cdot) \star W_3(c, z, \cdot, \cdot) .$$

Then the output of the third layer is given by $F_3 \in \mathbb{R}^{16\times 10\times 10}$ with

$$F_3(\cdot, i, j) = N_3(i, j)^{1/2} k \left(\sum_{m=1}^{6} \sum_{n,p=1}^{5} W_3(\cdot, m, n, p) W_3(\cdot, m, n, p)^T \right)^{-1/2} k \left(N_3(i, j)^{-1/2} F_3'(\cdot, i, j) \right)$$

for $i, j = 1, \ldots, 10$.

The fourth layer is similar to the second layer. The CKN pools and subsamples and then applies a kernel on 1×1 patches and projects onto a subspace. Define $\mathcal{E}_4 = (e_1, e_3, e_5, e_7)$ where $e_i \in \mathbb{R}^7$ is a vector with 1 in element i and 0 elsewhere. The pooling and subsampling result in $F_4' \in \mathbb{R}^{16\times 5\times 5}$ given by

$$F_4'(c, \cdot, \cdot) = \mathcal{E}_4^T \left(F_3(c, \cdot, \cdot) \star \frac{1}{4} \mathbb{1}_{2\times 2} \right) \mathcal{E}_4$$

for $c = 1, \ldots, 4$. Next, let $W_4 \in \mathbb{R}^{16\times 16}$ be the identity matrix and define N_4 to be a matrix containing the squared norms of 1×1 patches with a 1×1 stride

$$N_4 = \sum_{c=1}^{16} \left[F_4'(c, \cdot, \cdot) \odot F_4'(c, \cdot, \cdot) \right] .$$

Defining $F_4''(i, j, m) = F_4'(\cdot, i, j)^T W_4(m, \cdot)$ for all $i, j = 1, \ldots, 5$ and $m = 1, \ldots, 16$, the output of the fourth layer is then $F_4 \in \mathbb{R}^{16\times 5\times 5}$ given by

$$F_4(\cdot, i, j) = N_4(i, j)^{1/2} \left(\left[k \left(W_4(m, \cdot)^T W_4(n, \cdot) \right) \right]_{m,n=1}^{16} \right)^{-1/2} \left[k \left(N_4(i, j)^{-1/2} F_4''(i, j, m) \right) \right]_{m=1}^{16}$$

for $i, j = 1, \ldots, 5$.

The fifth layer is a fully connected layer. Let $W_5 \in \mathbb{R}^{120\times16\times5\times5}$. Then the output of this layer is given by $F_5 \in \mathbb{R}^{120}$ with

$$F_5 = \left(\left[k \left(\operatorname{vec}(W_5(m,\cdot,\cdot,\cdot))^T \operatorname{vec}(W_5(n,\cdot,\cdot,\cdot)) \right) \right]_{m,n=1}^{120} \right)^{-1/2} \times \left[k \left(\operatorname{vec}(F_4)^T \operatorname{vec}(W_5(m,\cdot,\cdot,\cdot)) \right) \right]_{m=1}^{120} .$$

Finally, the sixth layer is also a fully connected layer. Let $W_6 \in \mathbb{R}^{84\times120}$. Then the output is given by $F_6 \in \mathbb{R}^{84}$ with

$$F_6 = \left(\left[k \left(W_6(m,\cdot)^T W_6(n,\cdot) \right) \right]_{m,n=1}^{84} \right)^{-1/2} \left[k \left(F_5^T W_6(m,\cdot) \right) \right]_{m=1}^{84} .$$

The output from this layer is the set of features provided to a classifier.

A.3.2 All-CNN-C

For the CKN counterpart of All-CNN-C the input to the model is an image $F_0 \in \mathbb{R}^{3\times32\times32}$. Let $W_1 \in \mathbb{R}^{96\times3\times3\times3}$. The initial layer consists of projecting onto a subspace spanned by feature maps $\phi(\operatorname{vec}(W_{1,\cdot,\cdot,\cdot})), \ldots, \phi(\operatorname{vec}(W_{96,\cdot,\cdot,\cdot}))$ from the RBF kernel on the sphere on normalized patches and then multiplying by the norms of the patches. Define $Z_1 \in \mathbb{R}^{32\times34}$ such that $(Z_1)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 32$ and is 0 otherwise. Let N_1 be a matrix containing the squared norms of $3 \times 3 \times 3$ patches with a 1×1 stride:

$$N_1 = \sum_{c=1}^{3} \left\{ \left[(Z_1^T F_0(c,\cdot,\cdot) Z_1) \odot (Z_1^T F_0(c,\cdot,\cdot) Z_1) \right] \star \mathbb{1}_{3\times3} \right\},$$

where $\odot$ is the Hadamard product. Let $k : [-1, 1] \to \mathbb{R}$ be defined as $k(z) = \exp((z-1)/\sigma^2)$. Define

$$F_1'(c,\cdot,\cdot) = \sum_{z=1}^{3} \left[(Z_1^T F_0(z,\cdot,\cdot) Z_1) \star W_1(c,z,\cdot,\cdot) \right]$$

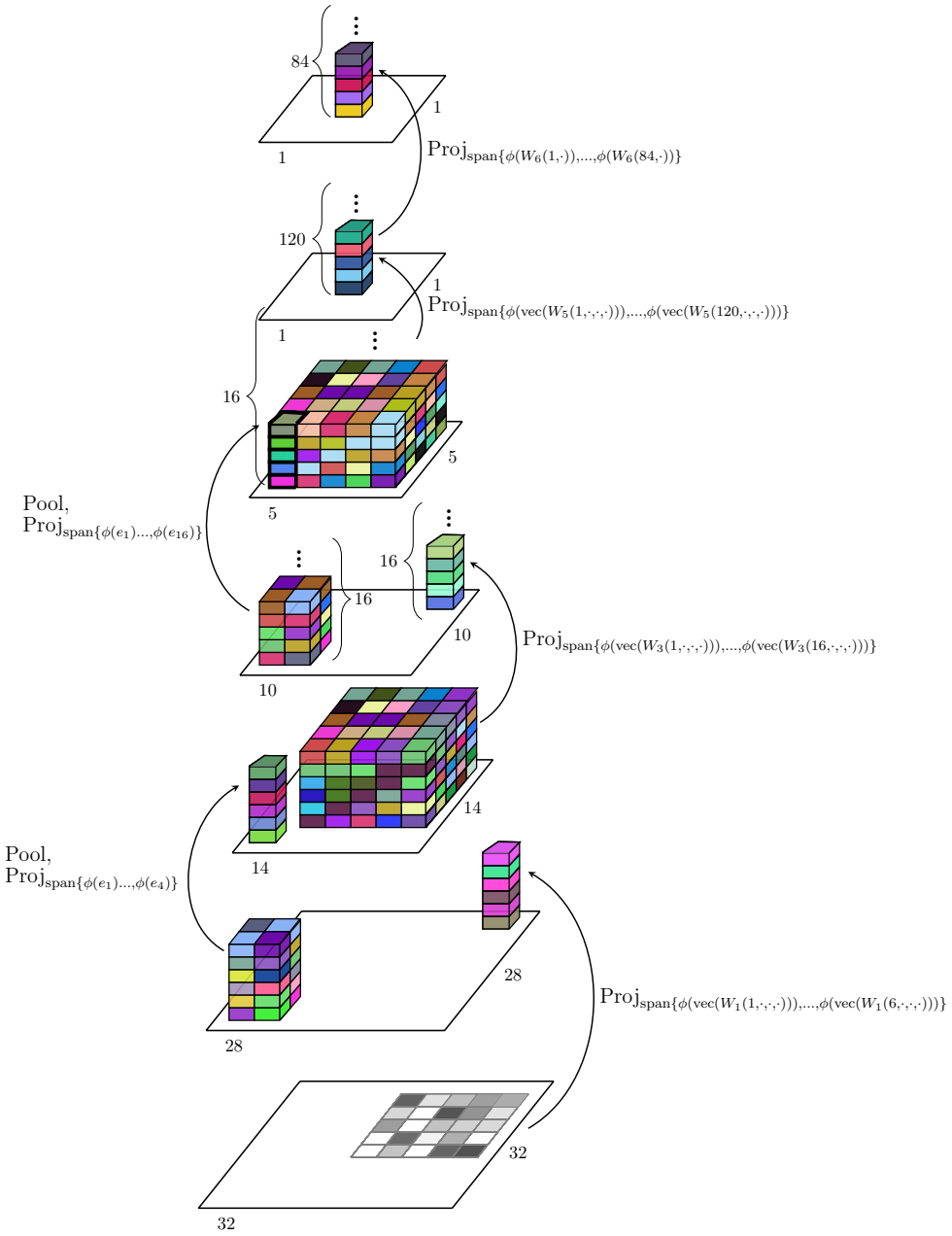


Figure A.3: LeNet-5 architecture. The numbers next to curly brackets indicate the number of filters at each layer when the number of filters is not the same as the height of the stack of blocks. See the caption of Figure A.2 for an explanation of the other components of the figure.

for $c = 1, \ldots, 96$. The output from the projection is then given by $F_1 \in \mathbb{R}^{96\times 32\times 32}$ with

$$F_1(\cdot, i, j) = N_1(i,j)^{1/2} k\left(\sum_{m=1}^{3}\sum_{n,p=1}^{3} W_1(\cdot, m, n, p) W_1(\cdot, m, n, p)^T\right)^{-1/2} k\left(N_1(i,j)^{-1/2} F_1'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 32$ where k is understood to be applied element-wise.

The second layer is of the same form as the first layer. Let $W_2 \in \mathbb{R}^{96\times 96\times 3\times 3}$ and define $Z_2 \in \mathbb{R}^{32\times 34}$ such that $(Z_2)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 32$ and is 0 otherwise. Let N_2 be a matrix containing the squared norms of $96 \times 3 \times 3$ patches with a 1×1 stride:

$$N_2 = \sum_{c=1}^{96} \left\{ \left[(Z_2^T F_1(c, \cdot, \cdot) Z_2) \odot (Z_2^T F_1(c, \cdot, \cdot) Z_2) \right] \star \mathbb{1}_{3\times 3} \right\} .$$

Define

$$F_2'(c, \cdot, \cdot) = \sum_{z=1}^{96} \left[(Z_2^T F_1(z, \cdot, \cdot) Z_2) \star W_2(c, z, \cdot, \cdot) \right]$$

for $c = 1, \ldots, 96$. The output from the projection is then given by $F_2 \in \mathbb{R}^{96\times 32\times 32}$ with

$$F_2(\cdot, i, j) = N_2(i,j)^{1/2} k\left(\sum_{m=1}^{96}\sum_{n,p=1}^{3} W_2(\cdot, m, n, p) W_2(\cdot, m, n, p)^T\right)^{-1/2} k\left(N_2(i,j)^{-1/2} F_2'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 32$.

The third layer is similar to the previous layer, but subsamples. Let $W_3 \in \mathbb{R}^{96\times 96\times 3\times 3}$ and define $\mathcal{E}_3 = (e_1, e_3, e_5, \ldots, e_{31})$ where $e_i \in \mathbb{R}^{32}$ is a vector with 1 in element i and 0 elsewhere. Let N_3 be a matrix containing the squared norms of $96 \times 3 \times 3$ patches with a 2×2 stride:

$$N_3 = \sum_{c=1}^{96} \mathcal{E}_3^T \left\{ [F_1(c, \cdot, \cdot) \odot F_1(c, \cdot, \cdot)] \star \mathbb{1}_{3\times 3} \right\} \mathcal{E}_3 .$$

Define

$$F_3'(c, \cdot, \cdot) = \sum_{z=1}^{96} \mathcal{E}_3^T \left[F_2(z, \cdot, \cdot) \star W_3(c, z, \cdot, \cdot) \right] \mathcal{E}_3$$

for $c = 1, \ldots, 96$. The output from the projection is then given by $F_3 \in \mathbb{R}^{96\times 15\times 15}$ with

$$F_3(\cdot, i, j) = N_3(i,j)^{1/2} k \left(\sum_{m=1}^{96} \sum_{n,p=1}^{3} W_3(\cdot, m, n, p) W_3(\cdot, m, n, p)^T \right)^{-1/2} k\left(N_3(i,j)^{-1/2} F_3'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 15$.

The fourth layer is of the same form as the second layer, but has 192 filters. Let $W_4 \in \mathbb{R}^{192\times 96\times 3\times 3}$ and define $Z_4 \in \mathbb{R}^{15\times 17}$ such that $(Z_4)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 15$ and is 0 otherwise. Let N_4 be a matrix containing the squared norms of $96 \times 3 \times 3$ patches with a 1×1 stride:

$$N_4 = \sum_{c=1}^{96} \left\{ \left[(Z_4^T F_3(c, \cdot, \cdot) Z_4) \odot (Z_4^T F_3(c, \cdot, \cdot) Z_4) \right] \star \mathbb{1}_{3\times 3} \right\} .$$

Define

$$F_4'(c, \cdot, \cdot) = \sum_{z=1}^{96} \left[(Z_4^T F_3(z, \cdot, \cdot) Z_4) \star W_4(c, z, \cdot, \cdot) \right]$$

for $c = 1, \ldots, 192$. The output from the projection is then given by $F_4 \in \mathbb{R}^{192\times 15\times 15}$ with

$$F_4(\cdot, i, j) = N_4(i,j)^{1/2} k \left(\sum_{m=1}^{96} \sum_{n,p=1}^{3} W_4(\cdot, m, n, p) W_4(\cdot, m, n, p)^T \right)^{-1/2} k\left(N_4(i,j)^{-1/2} F_4'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 15$.

The fifth layer is analogous to the fourth layer. Let $W_5 \in \mathbb{R}^{192\times 192\times 3\times 3}$ and define $Z_5 \in \mathbb{R}^{15\times 17}$ such that $(Z_5)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 15$ and is 0 otherwise. Let N_5 be a matrix containing the squared norms of $192 \times 3 \times 3$ patches with a 1×1 stride:

$$N_5 = \sum_{c=1}^{192} \left\{ \left[(Z_5^T F_4(c, \cdot, \cdot) Z_5) \odot (Z_5^T F_4(c, \cdot, \cdot) Z_5) \right] \star \mathbb{1}_{3\times 3} \right\} .$$

Define

$$F_5'(c, \cdot, \cdot) = \sum_{z=1}^{192} \left[(Z_5^T F_4(z, \cdot, \cdot) Z_5) \star W_5(c, z, \cdot, \cdot) \right]$$

for $c = 1, \ldots, 192$. The output from the projection is then given by $F_5 \in \mathbb{R}^{192\times 15\times 15}$ with

$$F_5(\cdot, i, j) = N_5(i,j)^{1/2} k \left(\sum_{m=1}^{192} \sum_{n,p=1}^{3} W_5(\cdot, m, n, p) W_5(\cdot, m, n, p)^T \right)^{-1/2} k\left(N_5(i,j)^{-1/2} F_5'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 15$.

The sixth layer is similar to the third layer. Let $W_6 \in \mathbb{R}^{192\times 192\times 3\times 3}$ and define $\mathcal{E}_6 = (e_1, e_3, e_5, \ldots, e_{13})$ where $e_i \in \mathbb{R}^{14}$ is a vector with 1 in element i and 0 elsewhere. Let N_6 be a matrix containing the squared norms of $192 \times 3 \times 3$ patches with a 2×2 stride:

$$N_6 = \sum_{c=1}^{192} \mathcal{E}_6^T \left\{ [F_5(c, \cdot, \cdot) \odot F_5(c, \cdot, \cdot)] \star \mathbb{1}_{3\times 3} \right\} \mathcal{E}_6 \, .$$

Define

$$F_6'(c, \cdot, \cdot) = \sum_{z=1}^{192} \mathcal{E}_6^T \left[F_5(z, \cdot, \cdot) \star W_6(c, z, \cdot, \cdot) \right] \mathcal{E}_6$$

for $c = 1, \ldots, 192$. The output from the projection is then given by $F_6 \in \mathbb{R}^{192\times 7\times 7}$ with

$$F_6(\cdot, i, j) = N_6(i,j)^{1/2} k \left(\sum_{m=1}^{192} \sum_{n,p=1}^{3} W_6(\cdot, m, n, p) W_6(\cdot, m, n, p)^T \right)^{-1/2} k\left(N_6(i,j)^{-1/2} F_6'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 7$.

The seventh layer is analogous to the fifth layer. Let $W_7 \in \mathbb{R}^{192\times 192\times 3\times 3}$ and define $Z_7 \in \mathbb{R}^{7\times 9}$ such that $(Z_7)_{ij} = 1$ if $j = i + 1$ for $i = 1, \ldots, 7$ and is 0 otherwise. Let N_7 be a matrix containing the squared norms of $192 \times 3 \times 3$ patches with a 1×1 stride:

$$N_7 = \sum_{c=1}^{192} \left\{ \left[(Z_7^T F_6(c, \cdot, \cdot) Z_7) \odot (Z_7^T F_6(c, \cdot, \cdot) Z_7) \right] \star \mathbb{1}_{3\times 3} \right\} .$$

Define

$$F_7'(c, \cdot, \cdot) = \sum_{z=1}^{192} \left[(Z_7^T F_6(z, \cdot, \cdot) Z_7) \star W_7(c, z, \cdot, \cdot) \right]$$

for $c = 1, \ldots, 192$. The output from the projection is then given by $F_7 \in \mathbb{R}^{96\times7\times7}$ with

$$F_7(\cdot, i, j) = N_7(i, j)^{1/2} k \left(\sum_{m=1}^{192} \sum_{n,p=1}^{3} W_7(\cdot, m, n, p) W_7(\cdot, m, n, p)^T \right)^{-1/2} k \left(N_7(i, j)^{-1/2} F_7'(\cdot, i, j) \right)$$

for $i, j = 1, \ldots, 7$.

The eighth layer switches to 1×1 convolutions. Let $W_8 \in \mathbb{R}^{192\times192}$ and let N_8 be a matrix containing the squared norms of $192 \times 1 \times 1$ patches with a 1×1 stride:

$$N_8 = \sum_{c=1}^{192} \left[F_7(c, \cdot, \cdot) \odot F_7(c, \cdot, \cdot) \right] .$$

Define

$$F_8'(c, \cdot, \cdot) = \sum_{z=1}^{192} \left[F_7(z, \cdot, \cdot) \star W_8(c, z) \right]$$

for $c = 1, \ldots, 192$. The output from the projection is then given by $F_8 \in \mathbb{R}^{192\times7\times7}$ with

$$F_8(\cdot, i, j) = N_8(i, j)^{1/2} k \left(\sum_{m=1}^{192} W_8(\cdot, m) W_8(\cdot, m)^T \right)^{-1/2} k \left(N_8(i, j)^{-1/2} F_8'(\cdot, i, j) \right)$$

for $i, j = 1, \ldots, 7$.

The ninth layer again has 1×1 convolutions, but with 10 filters. Let $W_9 \in \mathbb{R}^{10\times192}$ and let N_9 be a matrix containing the squared norms of $192 \times 1 \times 1$ patches with a 1×1 stride:

$$N_9 = \sum_{c=1}^{192} \left[F_8(c, \cdot, \cdot) \odot F_8(c, \cdot, \cdot) \right] .$$

Define

$$F_9'(c, \cdot, \cdot) = \sum_{z=1}^{192} \left[F_8(z, \cdot, \cdot) \star W_9(c, z) \right]$$

for $c = 1, \ldots, 10$. The output from the projection is then given by $F_9 \in \mathbb{R}^{10\times 7\times 7}$ with

$$F_9(\cdot, i, j) = N_9(i, j)^{1/2} k\left(\sum_{m=1}^{192} W_9(\cdot, m) W_9(\cdot, m)^T\right)^{-1/2} k\left(N_9(i, j)^{-1/2} F_9'(\cdot, i, j)\right)$$

for $i, j = 1, \ldots, 7$.

The tenth layer performs pooling across the entire feature maps. The output is given by $F_{10} \in \mathbb{R}^{10}$ with

$$F_{10}(c) = F_9(c, \cdot, \cdot) \star \frac{1}{49} \mathbb{1}_{7\times 7}$$

for $c = 1, \ldots, 10$. The output from this layer is the set of features provided to a classifier.

A.3.3 AlexNet

For the CKN counterpart of AlexNet we omit the CKN counterpart of the local response normalization layers since local response normalization provides little added benefit and has fallen out of favor in recent years (Simonyan and Zisserman, 2015; Krähenbühl et al., 2016).

Let $F_0 \in \mathbb{R}^{3\times 224\times 224}$ be the representation of an input image and $W_1 \in \mathbb{R}^{64\times 3\times 11\times 11}$. The initial layer consists of projecting onto a subspace spanned by the feature maps $\phi(\mathrm{vec}(W_{1,\cdot,\cdot,\cdot})), \ldots, \phi(\mathrm{vec}(W_{64,\cdot,\cdot,\cdot}))$ from the Gaussian RBF kernel on the sphere on normalized patches and then multiplying by the norms of the patches. It then pools and subsamples. Define the zero-padding matrix $Z_1 \in \mathbb{R}^{224\times 228}$ such that $(Z_1)_{ij} = 1$ if $j = i + 2$ for $i = 1, \ldots, 224$ and is 0 otherwise. Moreover, define $\mathcal{E}_1 = (e_1, e_5, e_9, \ldots, e_{217})$ where $e_i \in \mathbb{R}^{218}$ is a vector with 1 in element i and 0 elsewhere and define $\mathcal{E}_{1p} = (e_1, e_3, e_5, \ldots, e_{53})$ where $e_i \in \mathbb{R}^{53}$ is a vector with 1 in element i and 0 elsewhere. Let N_1 be a matrix containing the squared norms of $3 \times 11 \times 11$ patches with a 4×4 stride:

$$N_1 = \sum_{c=1}^{3} \mathcal{E}_1^T \left\{\left[\left(Z_1^T F_0(c, \cdot, \cdot) Z_1\right) \odot \left(Z_1^T F_0(c, \cdot, \cdot) Z_1\right)\right] \star \mathbb{1}_{11\times 11}\right\} \mathcal{E}_1$$

where $\odot$ is the Hadamard product. Let $k : [-1, 1] \to \mathbb{R}$ be defined as $k(z) = \exp((z - 1)/\sigma^2)$.

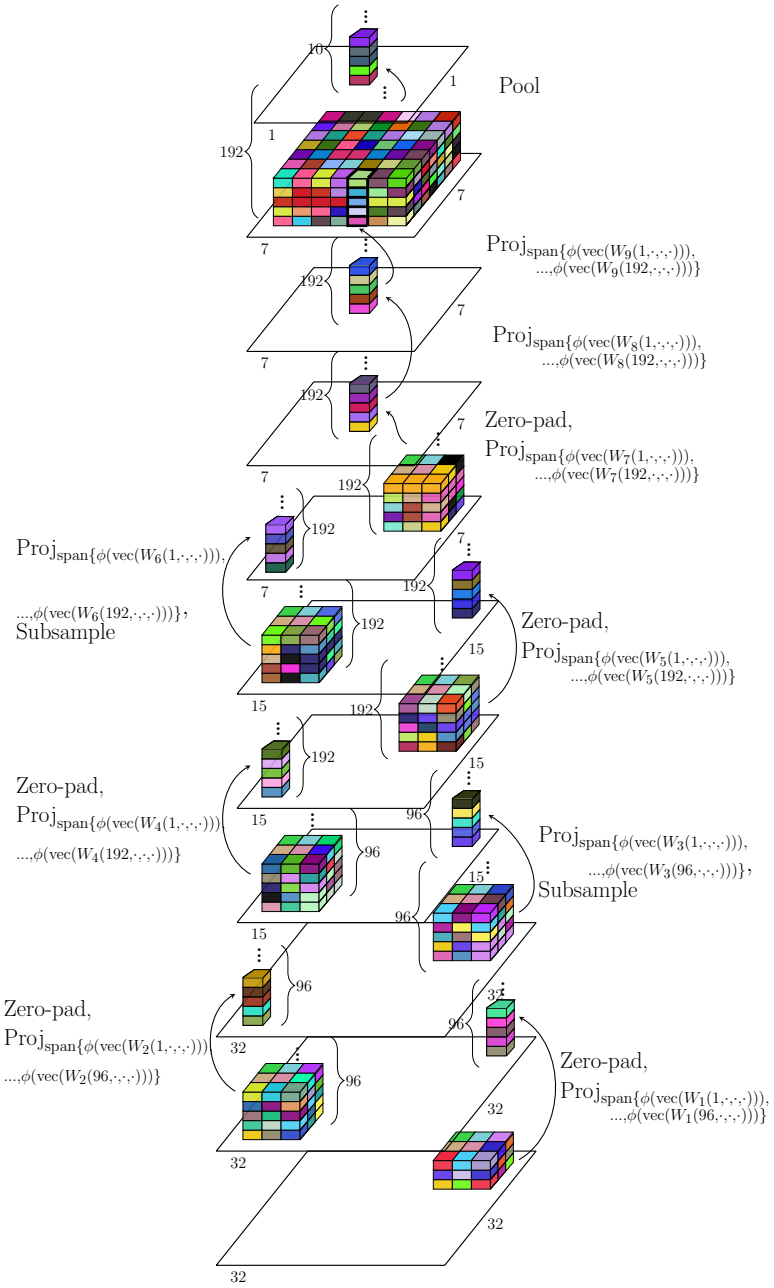


Figure A.4: All-CNN-C architecture. The numbers next to curly brackets indicate the number of filters at each layer when the number of filters is not the same as the height of the stack of blocks. See the caption of Figure A.2 for an explanation of the other components of the figure.

Define

$$F_1'(c,\cdot,\cdot) = \sum_{z=1}^{3} \mathcal{E}_1^T \left[\left(Z_1^T F_0(z,\cdot,\cdot) Z_1 \right) \star W_1(c,z,\cdot,\cdot) \right] \mathcal{E}_1$$

for $c = 1, \ldots 64$. The output from the projection is then given by

$$F_1''(\cdot,i,j) = N_1(i,j)^{1/2} k \left(\sum_{m=1}^{3} \sum_{n,p=1}^{11} W_1(\cdot,m,n,p) W_1(\cdot,m,n,p)^T \right)^{-1/2} k \left(N_1(i,j)^{-1/2} F_1'(\cdot,i,j) \right)$$

for $i,j = 1,\ldots,55$ where k is understood to be applied element-wise. The pooling and subsampling result in $F_1 \in \mathbb{R}^{64\times 27\times 27}$ given by

$$F_1(c,\cdot,\cdot) = \mathcal{E}_{1p}^T \left[\max_{i',j':i'\in\{i-1,i,i+1\},j'\in\{j-1,j,j+1\}} F_1''(c,i',j') \right]_{i,j=2}^{54} \mathcal{E}_{1p}$$

for $c = 1,\ldots,64$.

The second layer is similar to the first layer. Let $W_2 \in \mathbb{R}^{192\times 64\times 5\times 5}$ and define the zero-padding matrix $Z_2 \in \mathbb{R}^{27\times 31}$ such that $(Z_2)_{ij} = 1$ if $j = i+2$ for $i = 1,\ldots,27$ and is 0 otherwise. Moreover, define $\mathcal{E}_{2p} = (e_1, e_3, e_5, \ldots, e_{25})$ where $e_i \in \mathbb{R}^{25}$ is a vector with 1 in element i and 0 elsewhere. Let N_2 be a matrix containing the squared norms of $64\times 5\times 5$ patches with a 1×1 stride:

$$N_2 = \sum_{c=1}^{64} \left\{ \left[\left(Z_2^T F_1(c,\cdot,\cdot) Z_2 \right) \odot \left(Z_2^T F_1(c,\cdot,\cdot) Z_2 \right) \right] \star \mathbb{1}_{5\times 5} \right\} .$$

Define

$$F_2'(c,\cdot,\cdot) = \sum_{z=1}^{64} \left[\left(Z_2^T F_1(z,\cdot,\cdot) Z_2 \right) \star W_2(c,z,\cdot,\cdot) \right]$$

for $c = 1,\ldots 192$. The output from the projection is then given by

$$F_2''(\cdot,i,j) = N_2(i,j)^{1/2} k \left(\sum_{m=1}^{64} \sum_{n,p=1}^{5} W_2(\cdot,m,n,p) W_2(\cdot,m,n,p)^T \right)^{-1/2} k \left(N_2(i,j)^{-1/2} F_2'(\cdot,i,j) \right)$$

for $i, j = 1, \ldots, 25$. The pooling and subsampling result in $F_2 \in \mathbb{R}^{192\times 13\times 13}$ given by

$$F_2(c,\cdot,\cdot) = \mathcal{E}_{2p}^T \left[\max_{i',j':i'\in\{i-1,i,i+1\},j'\in\{j-1,j,j+1\}} F_2''(c,i',j') \right]_{i,j=2}^{26} \mathcal{E}_{2p}$$

for $c = 1, \ldots, 192$.

The third layer, unlike the previous two, has no pooling. Let $W_3 \in \mathbb{R}^{384\times 192\times 3\times 3}$ and define the zero-padding matrix $Z_3 \in \mathbb{R}^{13\times 15}$ such that $(Z_3)_{ij} = 1$ if $j = i+1$ for $i = 1, \ldots, 13$ and is 0 otherwise. Moreover, let N_3 be a matrix containing the squared norms of $192 \times 3 \times 3$ patches with a 1×1 stride:

$$N_3 = \sum_{c=1}^{192} \left\{ \left[\left(Z_3^T F_2(c,\cdot,\cdot) Z_3 \right) \odot \left(Z_3^T F_2(c,\cdot,\cdot) Z_3 \right) \right] \star \mathbb{1}_{3\times 3} \right\} .$$

Define

$$F_3'(c,\cdot,\cdot) = \sum_{z=1}^{192} \left[\left(Z_3^T F_2(z,\cdot,\cdot) Z_3 \right) \star W_3(c,z,\cdot,\cdot) \right]$$

for $c = 1, \ldots 384$. The output of the third layer is then $F_3 \in \mathbb{R}^{384\times 13\times 13}$ given by

$$F_3(\cdot,i,j) = N_3(i,j)^{1/2} k \left(\sum_{m=1}^{192} \sum_{n,p=1}^{3} W_3(\cdot,m,n,p) W_3(\cdot,m,n,p)^T \right)^{-1/2} k\left(N_3(i,j)^{-1/2} F_3'(\cdot,i,j) \right)$$

for $i, j = 1, \ldots, 13$.

The fourth layer is similar to the third layer. Let $W_4 \in \mathbb{R}^{256\times 384\times 3\times 3}$ and define the zero-padding matrix $Z_4 \in \mathbb{R}^{13\times 15}$ such that $(Z_4)_{ij} = 1$ if $j = i+1$ for $i = 1, \ldots, 13$ and is 0 otherwise. Moreover, let N_4 be a matrix containing the squared norms of $384 \times 3 \times 3$ patches with a 1×1 stride:

$$N_4 = \sum_{c=1}^{384} \left\{ \left[\left(Z_4^T F_3(c,\cdot,\cdot) Z_4 \right) \odot \left(Z_4^T F_3(c,\cdot,\cdot) Z_4 \right) \right] \star \mathbb{1}_{3\times 3} \right\} .$$

Define

$$F_4'(c,\cdot,\cdot) = \sum_{z=1}^{384} \left[\left(Z_4^T F_3(z,\cdot,\cdot) Z_4 \right) \star W_4(c,z,\cdot,\cdot) \right]$$

for $c = 1, \ldots 256$. The output of the fourth layer is then $F_4 \in \mathbb{R}^{256\times 13\times 13}$ given by

$$F_4(\cdot,i,j) = N_4(i,j)^{1/2} k \left(\sum_{m=1}^{384} \sum_{n,p=1}^{3} W_4(\cdot,m,n,p) W_4(\cdot,m,n,p)^T \right)^{-1/2} k\left(N_4(i,j)^{-1/2} F_4'(\cdot,i,j) \right)$$

for $i,j = 1,\ldots,13$.

The fifth layer reintroduces pooling and subsampling. Let $W_5 \in \mathbb{R}^{256\times 256\times 3\times 3}$ and define the zero-padding matrix $Z_5 \in \mathbb{R}^{13\times 15}$ such that $(Z_5)_{ij} = 1$ if $j = i+1$ for $i = 1,\ldots,13$ and is 0 otherwise. Moreover, define $\mathcal{E}_{5p} = (e_1, e_3, e_5, \ldots, e_{13})$ where $e_i \in \mathbb{R}^{13}$ is a vector with 1 in element i and 0 elsewhere. Let N_5 be a matrix containing the squared norms of $256 \times 3 \times 3$ patches with a 1×1 stride:

$$N_5 = \sum_{c=1}^{256} \left\{ \left[\left(Z_5^T F_4(c,\cdot,\cdot) Z_5 \right) \odot \left(Z_5^T F_4(c,\cdot,\cdot) Z_5 \right) \right] \star \mathbb{1}_{3\times 3} \right\} .$$

Define

$$F_5'(c,\cdot,\cdot) = \sum_{z=1}^{256} \left[\left(Z_5^T F_4(z,\cdot,\cdot) Z_5 \right) \star W_5(c,z,\cdot,\cdot) \right]$$

for $c = 1, \ldots 256$. The output from the projection is then given by

$$F_5''(\cdot,i,j) = N_5(i,j)^{1/2} k \left(\sum_{m=1}^{256} \sum_{n,p=1}^{3} W_5(\cdot,m,n,p) W_5(\cdot,m,n,p)^T \right)^{-1/2} k\left(N_5(i,j)^{-1/2} F_5'(\cdot,i,j) \right)$$

for $i,j = 1,\ldots,13$. The pooling and subsampling result in $F_5 \in \mathbb{R}^{256\times 6\times 6}$ given by

$$F_5(c,\cdot,\cdot) = \mathcal{E}_{5p}^T \left[\max_{i',j' : i' \in \{i-1,i,i+1\}, j' \in \{j-1,j,j+1\}} F_5''(c,i',j') \right]_{i,j=2}^{12} \mathcal{E}_{5p} ,$$

for $c = 1,\ldots,256$.

The last three layers are the analogues of fully connected layers. Let $W_6 \in \mathbb{R}^{4096\times 256\times 6\times 6}$

and define N_6 to be the squared norm of F_5:

$$N_6 = \|F_5\|_F^2 \ .$$

Define

$$F_6'(c) = \sum_{z=1}^{256} [F_5(z,\cdot,\cdot) \star W_6(c,z,\cdot,\cdot)]$$

for $c = 1, \ldots 4096$. The output of the sixth layer is then $F_6 \in \mathbb{R}^{4096}$ given by

$$F_6 = N_6^{1/2} k \left(\sum_{m=1}^{256} \sum_{n,p=1}^{6} W_6(\cdot,m,n,p) W_6(\cdot,m,n,p)^T \right)^{-1/2} k \left(N_6^{-1/2} F_6' \right) \ .$$

The seventh layer is similar. Let $W_7 \in \mathbb{R}^{4096 \times 4096}$ and define N_7 to be the squared norm of F_6:

$$N_7 = \|F_6\|_2^2 \ .$$

Define

$$F_7' = W_7 F_6 \ .$$

The output of the projection is then $F_7 \in \mathbb{R}^{4096}$ given by

$$F_7 = N_7^{1/2} k \left(\sum_{m=1}^{4096} W_7(\cdot,m) W_7(\cdot,m)^T \right)^{-1/2} k \left(N_7^{-1/2} F_7' \right) \ .$$

The output of this layer is then fed to a classifier.

A.4 Gradient of the Objective with Respect to the CKN Weight Matrices

In this section we provide the full derivation of the gradient of the loss function with respect to the CKN weight matrices, usually referred to as "filters" in the ConvNet terminology. Along the way we compute the gradient for a single layer of the CKN with respect to both the filters and the inputs. Following this, in the case when the kernels are Gaussian RBF kernels

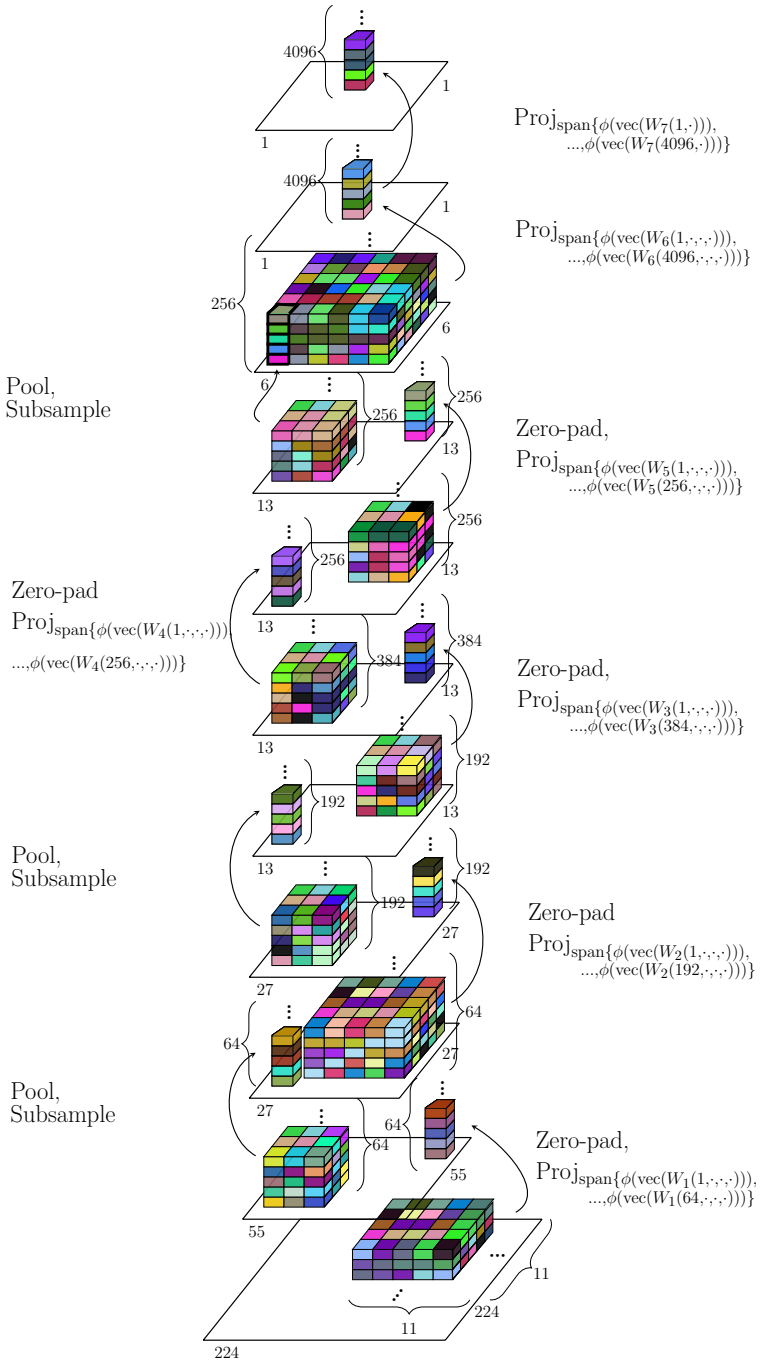


Figure A.5: AlexNet architecture. The numbers next to curly brackets indicate the number of filters at each layer when the number of filters is not the same as the height of the stack of blocks. See the caption of Figure A.2 for an explanation of the components of the figure.

on the sphere, we derive the gradient of the loss function with respect to the bandwidths of the kernels.

A.4.1 Notations

Linear algebra. We denote I_d the identity matrix in $\mathbb{R}^{d\times d}$. For a matrix $A \in \mathbb{R}^{d\times n} = (a_1, \dots, a_n)$, we denote $\mathrm{vec}(A) = (a_1; \dots; a_n) \in \mathbb{R}^{dn}$ the concatenation of its columns. Given two dimensions d, n, we denote $T_{d,n} \in \mathbb{R}^{dn\times dn}$ the linear operator such that for a matrix $A \in \mathbb{R}^{d\times n}$, $\mathrm{vec}(A^T) = T_{d,n}\,\mathrm{vec}(A)$. Given matrices $A_1, \dots, A_n$, we denote $\prod_{i=1}^n A_i = A_n A_{n-1} \dots A_1$, i.e., the multiplication is performed from right to left in increasing order of the indices.

We denote the set of all positive definite matrices of size $n \times n$ by S^n_{++}. We assume all matrices have real-valued entries and use the notation $A^{1/2}$ to denote the square root of a positive semi-definite matrix. That is, if $A = UDU^T$ is the eigendecomposition of A with U orthonormal and D diagonal, then $A^{1/2} = UD^{1/2}U^T$.

Derivatives. For $f : \mathbb{R}^d \to \mathbb{R}^n$, we denote by $\nabla f(x) = \left(\frac{\partial f_i(x)}{\partial x_j}\right)_{i=1,\dots,n,j=1,\dots,d} \in \mathbb{R}^{n\times d}$ its Jacobian at $x \in \mathbb{R}^d$, where $f_i(x)$ is the ith coordinate of $f(x)$. For a function $f : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}^n$, denoted $f(x, y)$ for $x \in \mathbb{R}^p$, $y \in \mathbb{R}^q$, we denote by $\nabla_x f(x, y)$ its partial Jacobian with respect to the variable x at a point $(x, y) \in \mathbb{R}^p \times \mathbb{R}^q$, i.e., $\nabla_x f(x, y) = \left(\frac{\partial f_i(x,y)}{\partial x_j}\right)_{i=1,\dots,n,j=1,\dots,p} \in \mathbb{R}^{n\times p}$ and similarly $\nabla_y f(x, y) = \left(\frac{\partial f_i(x,y)}{\partial y_j}\right)_{i=1,\dots,n,j=1,\dots,q} \in \mathbb{R}^{n\times q}$.

For a multivariate matrix function $\Psi : \mathbb{R}^{m\times n} \times \mathbb{R}^{p\times q} \to \mathbb{R}^{p^+\times q^+}$, denoted $\Psi(W, F) \in \mathbb{R}^{p^+\times q^+}$ for $W \in \mathbb{R}^{m\times n}$, $F \in \mathbb{R}^{p\times q}$, we denote by $\mathrm{vec}(\Psi) : \mathbb{R}^{mn} \times \mathbb{R}^{pq} \to \mathbb{R}^{p^+q^+}$ its vectorized counterpart such that $\mathrm{vec}(\Psi(W, F)) = \mathrm{vec}(\Psi)(\mathrm{vec}(W), \mathrm{vec}(F))$ for any $W \in \mathbb{R}^{m\times n}$, $F \in \mathbb{R}^{p\times q}$. We then denote by $\nabla_{\mathrm{vec}(W)} \mathrm{vec}(\Psi)(\mathrm{vec}(W), \mathrm{vec}(F)) \in \mathbb{R}^{p^+q^+\times mn}$ the partial Jacobian of the vectorized counterpart of Ψ with respect to its first argument $\mathrm{vec}(W)$, and analogously for $\nabla_{\mathrm{vec}(F)} \mathrm{vec}(\Psi)(\mathrm{vec}(W), \mathrm{vec}(F)) \in \mathbb{R}^{p^+q^+\times pq}$. In the following when W, F are clear from the context we denote simply $\nabla_{\mathrm{vec}(W)} \mathrm{vec}(\Psi) = \nabla_{\mathrm{vec}(W)} \mathrm{vec}(\Psi)(\mathrm{vec}(W), \mathrm{vec}(F))$ and $\nabla_{\mathrm{vec}(F)} \mathrm{vec}(\Psi) = \nabla_{\mathrm{vec}(F)} \mathrm{vec}(\Psi)(\mathrm{vec}(W), \mathrm{vec}(F))$.

A.4.2 Detailed description of a CKN layer

The output of a CKN layer was informally presented in Section 2.3.1. Now we precisely describe each component of a layer and its corresponding dimension.

At layers $\ell = 1, \ldots, L$, let f_ℓ be the number of filters, s_ℓ be the total size of the patches, p_ℓ be the number of patches prior to pooling, and p'_ℓ be the number of patches after pooling. Now at each layer ℓ let $F_\ell \in \mathbb{R}^{f_\ell \times p'_\ell}$ denote the features, $W_\ell \in \mathbb{R}^{f_\ell \times s_\ell}$ denote the filters, and $P_\ell \in \mathbb{R}^{p_\ell \times p'_\ell}$ denote the pooling matrix.

Define the *patch extraction function* at layer ℓ, $E_\ell : \mathbb{R}^{f_{\ell-1} \times p'_{\ell-1}} \to \mathbb{R}^{s_\ell \times p_\ell}$, by

$$E_\ell(X) = \sum_{i=1}^{s_\ell / f_{\ell-1}} E_{\ell 1 i} X E_{\ell 2 i} \tag{A.1}$$

with $E_{\ell 1 i} \in \mathbb{R}^{s_\ell \times f_{\ell-1}}$ and $E_{\ell 2 i} \in \mathbb{R}^{p'_{\ell-1} \times p_\ell}$ for $i = 1, \ldots, s_\ell / f_{\ell-1}$. This function takes as input the feature representation of a whole image from layer $\ell - 1$ with size $f_{\ell-1} \times p'_{\ell-1}$ and outputs its p_ℓ patches put in columns of size s_ℓ of a matrix $E_\ell(X) \in \mathbb{R}^{s_\ell \times p_\ell}$.

Define the *patch normalization* function, $\tilde{N}_\ell : \mathbb{R}^{s_\ell \times p_\ell} \to \mathbb{R}^{p_\ell \times p_\ell}$, by

$$\tilde{N}_\ell(X) = \left[(X^T X) \odot \mathrm{I}_{p_\ell} \right]^{1/2}. \tag{A.2}$$

This function outputs a square matrix whose diagonal contains the norms of the patches. Finally we denote by

$$N_\ell(X) = \tilde{N}_\ell(E_\ell(X))$$

the composition of the two previous operations.

The output of the convolutional kernel layer is then

$$F_\ell = \Psi_\ell(F_{\ell-1}, W_\ell) \coloneqq \left(k\left(W_\ell W_\ell^T\right) + \epsilon \, \mathrm{I}_{f_\ell} \right)^{-1/2} k\left(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1} \right) N_\ell(F_{\ell-1}) P_\ell, \tag{A.3}$$

where $\epsilon > 0$ and $k : \mathbb{R} \to \mathbb{R}$ is a differentiable dot product kernel understood to be applied

element-wise. In the following we denote it compactly by

$$\Psi_\ell(F_{\ell-1}, W_\ell) \coloneqq A(W_\ell)B(W_\ell, F_{\ell-1})N_\ell(F_{\ell-1})P_\ell,$$

where

$$\begin{aligned} A(W_\ell) &\coloneqq \left(k\left(W_\ell W_\ell^T\right) + \epsilon\, \mathrm{I}_{f_\ell}\right)^{-1/2} \\ B(W_\ell, F_{\ell-1}) &\coloneqq k\left(W_\ell E_\ell(F_{\ell-1})N_\ell(F_{\ell-1})^{-1}\right). \end{aligned} \tag{A.4}$$

Recall the objective on which we want to apply first order optimization schemes. Given a set of images $F_0^{(1)}, \ldots, F_0^{(n)}$ with corresponding labels $y^{(1)}, \ldots, y^{(n)}$, we consider a loss $\mathcal{L}$ with a linear classifier parameterized by W_{L+1}, leading to the optimization problem

$$\begin{aligned} \min_{W_1,\ldots,W_{L+1}} \quad & \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(y^{(i)}, \left\langle W_{L+1}, F_L^{(i)}\right\rangle\right) + \lambda\|W_{L+1}\|_F^2 \\ \text{subject to} \quad & W_\ell \in S^{d_\ell} \quad \text{for } \ell = 1, \ldots, L \end{aligned}$$

where $S^{d_\ell} = \prod_{j=1}^{f_\ell} \mathbb{S}^{s_\ell - 1}$ is the Cartesian product of Euclidean unit spheres in $\mathbb{R}^{s_\ell}$ and $F_L^{(i)}$ is the output of the Lth layer of the network described by (A.3) applied to the ith image. We henceforth assume that $\mathcal{L}$ is differentiable with respect to its second argument.

A.4.3 Full derivation of the gradient with respect to the CKN weight matrices

The gradient of the loss achieved by a CKN with respect to the weight matrices of the CKN is given by the chain rule, as recalled in the following proposition. The key elements are then the derivative of a layer with respect to its input and its weight matrix, which are detailed by a series of lemmas.

Proposition 21. *Let $\mathcal{L}(y, \langle W_{L+1}, F_L\rangle)$ be the loss incurred by an image-label sample (F_0, y), where F_L is the output of L layers of the network described by (2.3) and W_{L+1} parameterizes the linear classifier. Then the gradient of the loss with respect to the inner weights W_ℓ,*

$1 \leq \ell \leq L$ *is given by*

$$\nabla_{\mathrm{vec}(W_\ell)} \mathcal{L}\left(y, \langle W_{L+1}, F_L \rangle\right) = \mathcal{L}' \,\mathrm{vec}(W_{L+1})^T \left[\prod_{\ell'=\ell+1}^{L} \nabla_{\mathrm{vec}\left(F_{\ell'-1}\right)} \mathrm{vec}\left(\Psi_{\ell'}\right)\right] \nabla_{\mathrm{vec}(W_\ell)} \mathrm{vec}\left(\Psi_\ell\right),$$

where $\mathcal{L}' = \frac{\partial \mathcal{L}(\bar{y},\hat{y})}{\partial \hat{y}}|_{(\bar{y},\hat{y})=(y,\langle W_{L+1},F_L\rangle)}$, $\nabla_{\mathrm{vec}(F_{\ell'-1})} \mathrm{vec}(\Psi_{\ell'})$ *is detailed in Proposition 33, and* $\nabla_{\mathrm{vec}(W_\ell)} \mathrm{vec}(\Psi_\ell)$ *is detailed in Proposition 27.*

Layer derivative with respect to its weight matrix

The proof of the derivative of a CKN layer with respect to its weights is based on decomposing the gradient computations into the following lemmas.

Lemma 22. *Define the function* $F : S_{++}^n \to \mathbb{R}^{n\times n}$ *by* $F(A) = A^{1/2}$. *Then for a positive definite matrix* $A \in \mathbb{R}^{n\times n}$ *and a matrix* $H \in \mathbb{R}^{n\times n}$ *such that* $A + H$ *is positive definite we have*

$$\mathrm{vec}(F(A+H)) = \mathrm{vec}(F(A)) + \left(\mathrm{I}_n \otimes A^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \mathrm{vec}(H) + o(\|H\|_F).$$

Proof. First we will aim to find the matrix C such that

$$(A+H)^{1/2} = A^{1/2} + C.$$

To this end, observe that

$$\mathrm{vec}(H) = \left(\mathrm{I}_n \otimes (A+H)^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right) \mathrm{vec}(C).$$

The term $\left(\mathrm{I}_n \otimes (A+H)^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)$ is invertible. This follows from the fact that A and $A + H$ are positive definite, the eigenvalues of a Kronecker product are all products of the eigenvalues, and the sum of positive definite matrices is positive definite. Therefore, we obtain

$$\mathrm{vec}(C) = \left(\mathrm{I}_n \otimes (A+H)^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \mathrm{vec}(H).$$

Next, we will show that

$$\left(\mathrm{I}_n \otimes (A+H)^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} = \left(\mathrm{I}_n \otimes A^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} + O(\|H\|_F).$$

To see this, first note that

$$\begin{aligned}
&\left(\mathrm{I}_n \otimes (A+H)^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \\
&= \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \left[\left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \left(\mathrm{I}_n \otimes (A+H)^{1/2}\right) \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} + \mathrm{I}_n\right]^{-1} \\
&\quad \times \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \\
&= \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \Big[\mathrm{I}_n - \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \left(\mathrm{I}_n \otimes (A+H)^{1/2}\right) \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \\
&\quad + o\left(\left\| \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \left(\mathrm{I}_n \otimes (A+H)^{1/2}\right) \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \right\|_F\right)\Big] \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \\
&= \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1} - \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \left(\mathrm{I}_n \otimes (A+H)^{1/2}\right) \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \\
&\quad + o\left(\left\| \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \left(\mathrm{I}_n \otimes (A+H)^{1/2}\right) \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \right\|_F\right).
\end{aligned}$$

By Lemma 23, $(A+H)^{1/2} = A^{1/2} + O(\|H\|_F)$. Therefore,

$$\begin{aligned}
&\left(\mathrm{I}_n \otimes (A+H)^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \\
&= \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1} - \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \left(\mathrm{I}_n \otimes A^{1/2}\right) \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1} \\
&\quad + o\left(\left\| \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \left(\mathrm{I}_n \otimes A^{1/2}\right) \left(A^{1/2} \otimes \mathrm{I}_n\right)^{-1/2} \right\|_F\right) + O(\|H\|_F) \\
&= \left(\mathrm{I}_n \otimes A^{1/2} + A^{1/2} \otimes \mathrm{I}_n\right)^{-1} + O(\|H\|_F).
\end{aligned}$$

□

Lemma 23. *The matrix square root function $F : S^n_{++} \to \mathbb{R}^{n \times n}$ given by $F(A) = A^{1/2}$ is continuous.*

Proof. By 1/2-homogeneity of the square root function, it is sufficient to show that there

exists $C > 0$ such that for any $A, B \in S^n_{++}$,

$$\|A - B\|_2 \leq 1 \Rightarrow \|A^{1/2} - B^{1/2}\|_2 \leq C \tag{A.5}$$

where $\|\cdot\|_2$ denotes the operator norm associated with the Euclidean norm. Indeed, for any $A, B \in S^n_{++}$, denoting $\lambda = \|A - B\|_2$, $\tilde{A} = A/\lambda$, $\tilde{B} = B/\lambda$, we get $\|\tilde{A} - \tilde{B}\|_2 = 1$ and if (A.5) holds, then $\|\tilde{A}^{1/2} - \tilde{B}^{1/2}\|_2 \leq C$, which reads

$$\|A^{1/2} - B^{1/2}\|_2 \leq C\sqrt{\lambda} = C\|A - B\|_2^{1/2}.$$

1/2- Hölder continuity then implies continuity.

Now to prove (A.5), define

$$g(x) = \int_0^1 \left(1 - \frac{1}{1 + tx}\right) t^{-3/2} dt.$$

The function g can be extended as a function on positive definite matrices that acts on their spectra. Precisely, for $A \in S^n_{++}$, diagonalized as $A = U\Lambda U^\top$ with $U^{-1} = U^T$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, denote $g(A) = Ug(\Lambda)U^\top$ where $g(\Lambda) = \mathrm{diag}(g(\lambda_1), \ldots, g(\lambda_n))$.

We will show that there exists $K > 0$ such that

$$\|KA^{1/2} - g(A)\| \leq 2 \tag{A.6}$$

and that g is K'-Lipschitz-continuous on S^n_{++} for a given $K' > 0$. Then given $A, B \in S^n_{++}$ with $\|A - B\|_2 \leq 1$ we would get

$$\begin{aligned} \|A^{1/2} - B^{1/2}\|_2 &\leq \|A^{1/2} - g(A)/K\|_2 + \frac{1}{K}\|g(A) - g(B)\|_2 + \|B^{1/2} - g(B)/K\|_2 \\ &\leq \frac{4}{K} + \frac{K'}{K} := C. \end{aligned} \tag{A.7}$$

This gives (A.5) and concludes the proof.

For (A.6), after the change of variables $s = xt$, we get

$$\begin{aligned} g(x) &= x^{1/2} \int_0^x \frac{s}{1+s} s^{-3/2} ds \\ &= x^{1/2} \left(\int_0^\infty \frac{s}{1+s} s^{-3/2} ds - \int_x^\infty \frac{s}{1+s} s^{-3/2} ds \right) \\ &= K x^{1/2} + h(x), \end{aligned}$$

where $K = \int_0^\infty \frac{s}{1+s} s^{-3/2} ds = \int_0^1 \frac{s}{1+s} s^{-3/2} ds + \int_1^\infty \frac{s}{1+s} s^{-3/2} ds \leq \int_0^1 \frac{s}{1+s} s^{-3/2} ds + \int_1^\infty s^{-3/2} ds < \infty$ and $h(x) = -x^{\frac{1}{2}} \int_x^\infty \frac{s}{1+s} s^{-3/2} ds$ with $|h(x)| \leq x^{\frac{1}{2}} \int_x^\infty s^{-3/2} ds = 2$. Therefore we get (A.6), as

$$\|g(A) - KA^{1/2}\|_2 = \|h(A)\|_2 \leq 2$$

where $h(A)$ denotes the application of h on the spectrum of A. Boundedness of h on real numbers imply directly its boundedness on matrices. Now for the Lipschitz continuity of g, first note that the integral commutes with the matrix operations defining the diagonalization, such that

$$g(A) = \int_0^1 \mathrm{I}_n - (\mathrm{I}_n + tA)^{-1} t^{-3/2} dt.$$

Then

$$\begin{aligned} g(A) - g(B) &= \int_0^1 \left[(\mathrm{I}_n + tB)^{-1} - (\mathrm{I}_n + tA)^{-1} \right] t^{-3/2} dt \\ &= \int_0^1 (\mathrm{I}_n + tB)^{-1} (A - B) (\mathrm{I}_n + tA)^{-1} t^{-1/2} dt, \end{aligned}$$

where we used that $X^{-1} - Y^{-1} = X^{-1}(Y - X)Y^{-1}$. So finally

$$\begin{aligned} \|f(A) - f(B)\|_2 &= \left\| \int_0^1 (\mathrm{I}_n + tB)^{-1} (A - B) (\mathrm{I}_n + tA)^{-1} t^{-1/2} dt \right\|_2 \\ &\leq \int_0^1 \|(\mathrm{I}_n + tB)^{-1} (A - B) (\mathrm{I}_n + tA)^{-1} t^{-1/2}\|_2 dt \\ &\leq \int_0^1 \|(\mathrm{I}_n + tB)^{-1}\|_2 \|A - B\|_2 \|(\mathrm{I}_n + tA)^{-1}\|_2 t^{-1/2} dt \end{aligned}$$

$$\leq \|A - B\|_2 \int_0^1 t^{-1/2} dt = 2\|A - B\|_2,$$

which ensures (A.7) and therefore the $1/2$- Hölder continuity of the matrix square root function. □

The following three lemmas may be proven via Taylor expansions.

Lemma 24. *Define the function $F : S_{++}^n \to \mathbb{R}^{n\times n}$ by $F(A) = A^{-1}$. Then for a positive definite matrix $A \in \mathbb{R}^{n\times n}$ and a matrix $H \in \mathbb{R}^{n\times n}$ such that $A + H$ is positive definite we have*

$$F(A + H) = F(A) - F(A)HF(A) + o(\|H\|_F).$$

Lemma 25. *Define the function $F : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times m}$ by $F(A) = k\left(AA^T\right)$ where $k : \mathbb{R} \to \mathbb{R}$ is a differentiable dot product kernel computed element-wise on AA^T. Then for a matrix $A \in \mathbb{R}^{m\times n}$ we have*

$$F(A + H) = F(A) + k'\left(AA^T\right) \odot \left(HA^T + AH^T\right) + o\left(\|H\|_F\right),$$

where $\odot$ denotes the Hadamard (element-wise) product.

Lemma 26. *Let $B \in \mathbb{R}^{p\times n}$ and define the function $F : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times p}$ by $F(A) = k\left(AB^T\right)$ where $k : \mathbb{R} \to \mathbb{R}$ is a differentiable dot product kernel computed element-wise on AB^T. Then for a matrix $A \in \mathbb{R}^{m\times n}$ we have*

$$F(A + H) = F(A) + k'\left(AB^T\right) \odot \left(HB^T\right) + o(\|H\|_F).$$

Now we are ready to differentiate a layer of a CKN with respect to its weight matrix.

Proposition 27 (Derivative of the CKN layer with respect to the weights)**.** *The output of the ℓth convolutional kernel layer defined in* (A.3)*,*

$$\Psi_\ell(F_{\ell-1}, W_\ell) = \left(k\left(W_\ell W_\ell^T\right) + \epsilon\, \mathrm{I}_{f_\ell}\right)^{-1/2} k\left(W_\ell E_\ell(F_{\ell-1}) N_\ell^{-1}(F_{\ell-1})\right) N_\ell(F_{\ell-1}) P_\ell,$$

$$= A(W_\ell)B(W_\ell, F_{\ell-1})N_\ell(F_{\ell-1})P_\ell$$

where $A(W_\ell), B(W_\ell, F_{\ell-1})$ *are defined in* (A.4)*, has a partial derivative with respect to the weights given by*

$$\begin{aligned}
\nabla_{\mathrm{vec}(W_\ell)} \mathrm{vec}(\Psi_\ell) = &- \left[(B(W_\ell, F_{\ell-1})N_\ell(F_{\ell-1})P_\ell)^T \otimes \mathrm{I}_{f_\ell}\right] \\
&\times (\mathrm{I}_{f_\ell} \otimes A(W_\ell) + A(W_\ell) \otimes \mathrm{I}_{f_\ell})^{-1} \left(A(W_\ell)^2 \otimes A(W_\ell)^2\right) \\
&\times \mathrm{diag}\left[\mathrm{vec}\left(k'\left(W_\ell W_\ell^T\right)\right)\right] (W_\ell \otimes \mathrm{I}_{f_\ell} + (\mathrm{I}_{f_\ell} \otimes W_\ell) T_{f_\ell, s_\ell}) \\
&+ \left[(N_\ell(F_{\ell-1})P_\ell)^T \otimes A(W_\ell)\right] \\
&\times \mathrm{diag}\left[\mathrm{vec}\left(k'\left(W_\ell E_\ell(F_{\ell-1})N_\ell(F_{\ell-1})^{-1}\right)\right)\right] \\
&\times \left[(E_\ell(F_{\ell-1})N_\ell(F_{\ell-1})^{-1})^T \otimes \mathrm{I}_{f_\ell}\right],
\end{aligned}$$

where T_{f_ℓ, s_ℓ} *is defined in Section A.4.1.*

Proof. We drop the layer index and simply denote $W = W_\ell$ and $F = F_{\ell-1}$. In addition, we denote $N = N(F)$ and $E = E(F)$. Denote the Fréchet derivative of a function f at a point A in the direction H by $L_f(A, H)$. On fixed inputs F, denote $\tilde{\Psi}(W) = \Psi(W, F)$ the restricted output function. Similarly we denote $\tilde{B}(W) = B(W, F)NP$. Observe that

$$\mathrm{vec}(\tilde{\Psi}(W)) = (\tilde{B}(W)^T \otimes \mathrm{I}_f)\,\mathrm{vec}(A(W)).$$

Therefore, we have by the product rule and chain rule (Higham, 2008, Theorems 3.3, 3.4) that

$$L_{\mathrm{vec}(\tilde{\Psi})}(W, H) = L_{\tilde{B}^T \otimes \mathrm{I}_f}(W, H)\,\mathrm{vec}(A(W)) + (\tilde{B}(W)^T \otimes \mathrm{I}_f) L_{\mathrm{vec}(A)}(W, H).$$

By the chain rule and Lemma 26, we have that

$$L_{\tilde{B}^T \otimes \mathrm{I}_f}(W, H)\,\mathrm{vec}(A(W))$$

$$
\begin{aligned}
&= \left\{ \left\{ \left[k'\left(WEN^{-1}\right) \odot \left(HEN^{-1}\right) \right] NP \right\}^T \otimes \mathrm{I}_f \right\} \mathrm{vec}(A(W)) \\
&= (P^T N^T \otimes A(W)) \,\mathrm{diag}\left[\mathrm{vec}\left(k'\left(WEN^{-1}\right) \right) \right] \left((EN^{-1})^T \otimes \mathrm{I}_f \right) \mathrm{vec}\left(H\right).
\end{aligned}
$$

Additionally, by the chain rule, and Lemmas 22 - 25 we have

$$
\begin{aligned}
(\tilde{B}(W)^T \otimes \mathrm{I}_f) L_{\mathrm{vec}(A)}(W,H) = &- \left(\tilde{B}(W)^T \otimes \mathrm{I}_f\right) \left(\mathrm{I}_f \otimes A(W) + A(W) \otimes \mathrm{I}_f\right)^{-1} \\
&\times \mathrm{vec}\left\{ A(W)^2 \left[k'\left(WW^T\right) \odot \left(HW^T + WH^T\right) \right] A(W)^2 \right\} \\
= &- \left(\tilde{B}(W)^T \otimes \mathrm{I}_f\right) \left(\mathrm{I}_f \otimes A(W) + A(W) \otimes \mathrm{I}_f\right)^{-1} \\
&\times \left(A(W)^2 \otimes A(W)^2 \right) \\
&\times \mathrm{diag}\left[\mathrm{vec}\left(k'\left(WW^T\right) \right) \right] \left(W \otimes \mathrm{I}_f + (\mathrm{I}_f \otimes W) T^T \right) \mathrm{vec}(H),
\end{aligned}
$$

where $T_{f,s}$ is the vectorized transpose matrix satisfying $\mathrm{vec}(H^T) = T_{f,s}\,\mathrm{vec}(H)$ for $H \in \mathbb{R}^{f \times s}$. Therefore,

$$
\begin{aligned}
\nabla_{\mathrm{vec}(W)} \mathrm{vec}(\Psi) = &- \left[\tilde{B}(W)^T \otimes \mathrm{I}_f\right] \left(\mathrm{I}_f \otimes A(W) + A(W) \otimes \mathrm{I}_f\right)^{-1} \left(A(W)^2 \otimes A(W)^2 \right) \\
&\times \mathrm{diag}\left[\mathrm{vec}\left(k'\left(WW^T\right) \right) \right] \left(W \otimes \mathrm{I}_f + (\mathrm{I}_f \otimes W) T_{f,s} \right) \\
&+ (P^T N \otimes A(W)) \,\mathrm{diag}\left[\mathrm{vec}\left(k'\left(WEN^{-1}\right) \right) \right] \left((EN^{-1})^T \otimes \mathrm{I}_f \right).
\end{aligned}
$$

□

Layer derivative with respect to its input

The proof of the derivative of a CKN layer with respect to its inputs is based on decomposing the gradient computations into Lemma 26 and the following additional lemmas. Lemmas 29 and 31 may be proven via Taylor expansions.

Lemma 28. *Let $M_1 \in \mathbb{R}^{m \times n}$ and $M_2 \in \mathbb{R}^{p \times q}$ and define the function $F : \mathbb{R}^{n \times p} \times \mathbb{R}^{m \times q}$ by*

$F(A) = M_1 A M_2$. Then for $H \in \mathbb{R}^{n\times p}$ we have

$$F(A+H) = F(A) + F(H).$$

Lemma 29. *Define the function $F : \mathbb{R}^d\backslash\{0\} \to \mathbb{R}$ by $F(x) = \|x\|$. Then for $x \in \mathbb{R}^d\backslash\{0\}$ and $h \in \mathbb{R}^d$ we have*

$$F(x+h) = F(x) + F^{-1}(x)x^T h + o(\|h\|).$$

Corollary 30. *Define the function $F : \mathbb{R}^{m\times n} \to \mathbb{R}^{n\times n}$ by $F(A) = \left[(A^T A) \odot \mathrm{I}_n\right]^{1/2}$. Then for $A \in \mathbb{R}^{m\times n}$ with $A_{\cdot,j} \in \mathbb{R}^m\backslash\{0\}$ for all $j = 1,\ldots,n$ and $H \in \mathbb{R}^{m\times n}$ we have*

$$F(A+H) = F(A) + F^{-1}(A) \odot (A^T H) + o(\|H\|_F).$$

Lemma 31. *Define the function $F : \mathbb{R}^d\backslash\{0\} \to \mathbb{R}$ by $F(x) = \|x\|^{-1}$. Then for $x \in \mathbb{R}^d\backslash\{0\}$ and $h \in \mathbb{R}^d$ we have*

$$F(x+h) = F(x) - F^3(x)x^T h + o(\|h\|).$$

Corollary 32. *Define the function $F : \mathbb{R}^{m\times n} \to \mathbb{R}^{n\times n}$ by $F(A) = \left[(A^T A) \odot \mathrm{I}_n\right]^{-1/2}$. Then for $A \in \mathbb{R}^{m\times n}$ with $A_{\cdot,j} \in \mathbb{R}^m\backslash\{0\}$ for all $j = 1,\ldots,n$ and $H \in \mathbb{R}^{m\times n}$ we have*

$$F(A+H) = F(A) - F(A)^{-3} \odot (A^T H) + o(\|H\|_F).$$

Proposition 33 (Derivative of the network with respect to the input)**.** *The output of the ℓth convolutional kernel layer defined in* (A.3)*,*

$$\begin{aligned}\Psi_\ell(F_{\ell-1}, W_\ell) &= \left(k\left(W_\ell W_\ell^T\right) + \epsilon\,\mathrm{I}_{f_\ell}\right)^{-1/2} k\left(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1}\right) N_\ell(F_{\ell-1}) P_\ell, \\ &= A(W_\ell) B(W_\ell, F_{\ell-1}) N_\ell(F_{\ell-1}) P_\ell\end{aligned}$$

where $A(W_\ell), B(W_\ell, F_{\ell-1})$ *are defined in* (A.4)*, has a partial derivative with respect to the inputs given by*

$$
\begin{aligned}
&\nabla_{\mathrm{vec}(F_{\ell-1})} \mathrm{vec}(\Psi_\ell) \\
=&\Bigg\{ \Bigg(\left[P_\ell^T N_\ell(F_{\ell-1}) \otimes A(W_\ell)\right] \mathrm{diag}\left[\mathrm{vec}\left(k'(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1}\right)\right] \\
&\quad \times \Big[\left[N_\ell(F_{\ell-1})^{-1} \otimes W_\ell\right] \\
&\qquad - \left[\mathrm{I}_{p_\ell} \otimes (W_\ell E_\ell(F_{\ell-1}))\right] \mathrm{diag}\left[\mathrm{vec}\left(N_\ell(F_{\ell-1})^{-3}\right)\right] \times \left[\mathrm{I}_{p_\ell} \otimes E_\ell(F_{\ell-1})^T\right] \Big] \Bigg) \\
&+ \left[P_\ell^T \otimes (A(W_\ell) B(W_\ell, F_{\ell-1}))\right] \mathrm{diag}\left[\mathrm{vec}\left(N_\ell(F_{\ell-1})^{-1}\right)\right] \times \left[\mathrm{I}_{p_\ell} \otimes E_\ell(F_{\ell-1})^T\right] \Bigg\} \times \Sigma_\ell.
\end{aligned}
$$

where $\Sigma_\ell = \sum_{i=1}^{s_\ell / f_{\ell-1}} \left(E_{\ell 2i}^T \otimes E_{\ell 1i}\right)$.

Proof. For simplicity in the proof we drop the layer index, denoting e.g., $W = W_\ell$ for the weights and $F = F_{\ell-1}$ for the input of this layer. Denote the Fréchet derivative of a function f at a point A in the direction H by $L_f(A, H)$. For fixed weights W, denote by $\tilde{\Psi}(F) = \Psi(W, F)$ the restricted output function. Recall that $N(F) = \tilde{N}(E(F))$ where $\tilde{N}$ and E are defined respectively in (A.1), (A.2). Furthermore, define $\Sigma \coloneqq \sum_{i=1}^{s/f} \left(E_{\ell 2i}^T \otimes E_{\ell 1i}\right)$. We have by the product rule and chain rule (Higham, 2008, Theorems 3.3, 3.4) that

$$
\begin{aligned}
L_{\mathrm{vec}(\tilde{\Psi})}(F, H) = &\mathrm{vec}\left(A(W) L_k\left(WE(F)N(F)^{-1}, \right.\right. \\
&\left.\left. WL_E(F, H)N(F)^{-1} + WE(F) L_{\tilde{N}^{-1}}(E(F), L_E(F, H))\right) N(F) P\right) \\
&+ \mathrm{vec}\left(A(W) B(W, F) L_{\tilde{N}}(E(F), L_E(F, H)) P\right).
\end{aligned} \tag{A.8}
$$

Now consider the first of the two terms in equation (A.8). By Lemmas 26 and 28 and Corollary 32 we have

$$
\mathrm{vec}\left(A(W) L_k\left(WE(F)N(F)^{-1}, \right.\right.
$$

$$
\begin{aligned}
&WL_E(F,H)N(F)^{-1} + WE(F)L_{\tilde{N}^{-1}}(E(F), L_E(F,H))\big)\,\tilde{N}(E(F))P\Big) \\
&= \operatorname{vec}\Big\{A(W)\Big\{k'(WE(F)N(F)^{-1}) \\
&\quad \odot \Big[WE(H)N(F)^{-1} - WE(F)(\tilde{N}^{-3}(E(F)) \odot (E(F)^T E(H)))\Big]\Big\}\,\tilde{N}(E(F))P\Big\} \\
&= \big[(N(F)P)^T \otimes A(W)\big]\operatorname{diag}\big[\operatorname{vec}\big(k'(WE(F)N(F)^{-1})\big)\big] \\
&\quad \times \Big\{\big((N(F)^{-1})^T \otimes W\big) - [\mathrm{I}_p \otimes (WE(F))]\operatorname{diag}\big(\operatorname{vec}\big[N(F)^{-3}\big]\big)\,\big(\mathrm{I}_p \otimes E(F)^T\big)\Big\}\,\Sigma\operatorname{vec}(H)\,.
\end{aligned}
$$

Now consider the second term in (A.8). By Lemma 28 and Corollary 30 we have

$$
\begin{aligned}
&\operatorname{vec}\left(A(W)B(W,F)L_{\tilde{N}}(E(F), L_E(F,H))P\right) \\
&= \operatorname{vec}\left(A(W)B(W,F)\left[\tilde{N}(E(F))^{-1} \odot \big(E(F)^T E(H)\big)\right]P\right) \\
&= \big[P^T \otimes (A(W)B(W,F))\big]\operatorname{diag}\big[\operatorname{vec}(N(F)^{-1})\big]\,\big(\mathrm{I}_p \otimes E(F)^T\big)\,\Sigma\operatorname{vec}(H).
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
L_{\operatorname{vec}(\tilde{\Psi})}(F,H) = \Big\{ &\big[(N(F)P)^T \otimes A(W)\big]\operatorname{diag}\big[\operatorname{vec}\big(k'(WE(F)N(F)^{-1})\big)\big] \\
&\times \Big\{\big[N(F)^{-1} \otimes W\big] - [\mathrm{I}_p \otimes (WE(F))]\operatorname{diag}\big(\operatorname{vec}\big[N(F)^{-3}\big]\big)\,\big(\mathrm{I}_p \otimes E(F)^T\big)\Big\} \\
&+ \big[P^T \otimes A(W)B(W,F)\big]\operatorname{diag}\big[\operatorname{vec}(N(F)^{-1})\big]\,\big(\mathrm{I}_p \otimes E(F)^T\big)\Big\}\Sigma\operatorname{vec}(H).
\end{aligned}
$$

□

A.4.4 Derivative with respect to the bandwidth parameters

We compute here the derivatives with respect to the bandwidths when using Gaussian RBF kernels on the sphere. We write the Gaussian RBF kernel for x and x' such that $\|x\|_2 = \|x'\|_2 = 1$ by

$$
k_\sigma(x,x') = \exp\left(-\frac{\|x-x'\|_2^2}{2\sigma^2}\right) = \exp\left(-\frac{1-\langle x,x'\rangle}{\sigma^2}\right),
$$

where σ is the kernel bandwidth. The output of the convolutional kernel layer incorporates the bandwidth as

$$\begin{aligned}&\Psi_\ell(F_{\ell-1}, W_\ell, \sigma_\ell) \coloneqq \\ &\left(k_{\sigma_\ell}\left(W_\ell W_\ell^T\right) + \epsilon\, \mathrm{I}_{f_\ell}\right)^{-1/2} k_{\sigma_\ell}\left(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1}\right) N_\ell(F_{\ell-1}) P_\ell,\end{aligned} \tag{A.9}$$

where k_{σ_ℓ} is understood to be applied element-wise. First note that the gradient back-propagates through the network as in Proposition 2. This is stated in the following proposition.

Proposition 34. *Let $\mathcal{L}(y, \langle W_{L+1}, F_L\rangle)$ be the loss incurred by an image-label sample (F_0, y), where F_L is the output of L layers of the network described by* (A.9) *and W_{L+1} parameterizes the linear classifier. Then the gradient of the loss with respect to the bandwidths σ_ℓ, $1 \leq \ell \leq L$, is given by*

$$\nabla_{\sigma_\ell} \mathcal{L}\left(y, \langle W_{L+1}, F_L\rangle\right) = \mathcal{L}' \operatorname{vec}(W_{L+1})^T \left[\prod_{\ell'=\ell+1}^{L} \nabla_{\operatorname{vec}\left(F_{\ell'-1}\right)} \operatorname{vec}\left(\Psi_{\ell'}\right)\right] \nabla_{\sigma_\ell} \operatorname{vec}\left(\Psi_\ell\right),$$

where $\mathcal{L}' = \frac{\partial \mathcal{L}(\bar{y},\hat{y})}{\partial \hat{y}}|_{(\bar{y},\hat{y})=(y,\langle W_{L+1}, F_L\rangle)}$, $\nabla_{\operatorname{vec}(F_{\ell'-1})} \operatorname{vec}(\Psi_{\ell'})$ is detailed in Proposition 33, and $\nabla_{\sigma_\ell} \operatorname{vec}\left(\Psi_\ell\right)$ is detailed in Proposition 36.

The derivative of the Gaussian RBF kernel on the sphere with respect to its bandwidth is given by the following lemma. This leads us to the derivative of the network with respect to the bandwidth in the proposition below.

Lemma 35. *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times n}$ and define the function $F : \mathbb{R} \to \mathbb{R}^{m\times p}$ by $F(\sigma) = \exp\left[-\frac{1}{\sigma^2}(\mathbb{1}_{m\times p} - AB^T)\right]$, where $\mathbb{1}_{m\times p}$ is an $m\times p$ matrix of ones and* exp *is understood to be applied element-wise. Then for a scalar $h \in \mathbb{R}$ we have*

$$F(\sigma + h) = F(\sigma) + \frac{2}{\sigma^3} F(\sigma) \odot \left(\mathbb{1}_{m\times p} - AB^T\right) h + o(h). \tag{A.10}$$

Proposition 36 (Derivative of the network with respect to the bandwidth)**.** *The output of the ℓth convolutional kernel layer defined in* (A.9)*,*

$$\begin{aligned}\Psi_\ell(F_{\ell-1}, W_\ell, \sigma_\ell) &= \left(k_{\sigma_\ell}\left(W_\ell W_\ell^T\right) + \epsilon\, \mathrm{I}_{f_\ell}\right)^{-1/2} k_{\sigma_\ell}\left(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1}\right) N_\ell(F_{\ell-1}) P_\ell, \\ &= A_{\sigma_\ell}(W_\ell) B_{\sigma_\ell}(W_\ell, F_{\ell-1}) N_\ell(F_{\ell-1}) P_\ell,\end{aligned}$$

where $A_{\sigma_\ell}(W_\ell), B_{\sigma_\ell}(W_\ell, F_{\ell-1})$ are defined as in (A.4)*, has a partial derivative with respect to the bandwidth of the kernel given by*

$$\begin{aligned}\nabla_{\sigma_\ell} \operatorname{vec}(\Psi_\ell) = &- \left[(B_{\sigma_\ell}(W_\ell, F_{\ell-1}) N_\ell(F_{\ell-1}) P_\ell)^T \otimes \mathrm{I}_f\right] \left(\mathrm{I}_f \otimes A_{\sigma_\ell}(W_\ell) + A_{\sigma_\ell}(W_\ell) \otimes \mathrm{I}_f\right)^{-1} \\ &\times \left(A_{\sigma_\ell}(W_\ell)^2 \otimes A_{\sigma_\ell}(W_\ell)^2\right) \operatorname{vec}\left(\frac{2}{{\sigma_\ell}^3} k_{\sigma_\ell}\left(W_\ell W_\ell^T\right) \odot \left(\mathbb{1}_{f_\ell \times f_\ell} - W_\ell W_\ell^T\right)\right) \\ &+ \left(P_\ell^T N_\ell(F_{\ell-1}) \otimes A_{\sigma_\ell}(W_\ell)\right) \\ &\times \operatorname{vec}\left(\frac{2}{{\sigma_\ell}^3} B_{\sigma_\ell}(W_\ell, F_{\ell-1}) \odot \left(\mathbb{1}_{f_\ell \times p'_\ell} - W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1}\right)\right).\end{aligned}$$

Proof. For simplicity in the proof we drop the layer index, denoting e.g., $W = W_\ell$ for the weights and $F = F_{\ell-1}$ for the input of this layer. Denote the Fréchet derivative of a function f at a point σ in the direction h by $L_f(\sigma, h)$. For fixed weights W and inputs F, denote $\tilde{\Psi}(\sigma) = \Psi(W, F, \sigma)$ the restricted output function. Similarly denote $\tilde{A}(\sigma) = A_\sigma(W)$ and $\tilde{B}(\sigma) = B_\sigma(W, F) N(F) P$.

As in Proposition 27, observe that

$$\operatorname{vec}(\tilde{\Psi}(\sigma)) = (\tilde{B}(\sigma)^T \otimes \mathrm{I}_f) \operatorname{vec}(\tilde{A}(\sigma)).$$

By the chain rule and Lemmas 22, 24, and 35 we have

$$\begin{aligned}(\tilde{B}(\sigma)^T \otimes \mathrm{I}_f) L_{\operatorname{vec}(\tilde{A})}(\sigma, h) = &- (\tilde{B}(\sigma)^T \otimes \mathrm{I}_f) \left(\mathrm{I}_f \otimes \tilde{A}(\sigma) + \tilde{A}(\sigma) \otimes \mathrm{I}_f\right)^{-1} \left(\tilde{A}(\sigma)^2 \otimes \tilde{A}(\sigma)^2\right) \\ &\times \operatorname{vec}\left(\frac{2}{\sigma^3} k_\sigma\left(WW^T\right) \odot \left(\mathbb{1}_{f \times f} - WW^T\right)\right) h.\end{aligned}$$

Additionally, by the chain rule, Lemma 26, and Lemma 24 we have that

$$
\begin{aligned}
&L_{\tilde{B}^T \otimes \mathrm{I}_f}(\sigma, h)\,\mathrm{vec}(\tilde{A}(\sigma)) \\
&= \left[P^T N(F) \otimes \tilde{A}(\sigma)\right] \mathrm{vec}\left(\frac{2}{\sigma^3} k_\sigma\left(WE(F)N(F)^{-1}\right) \odot \left(\mathbb{1}_{f\times p'} - WE(F)N(F)^{-1}\right)\right) h.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\nabla_\sigma \,\mathrm{vec}(\Psi) = &- \left[\tilde{B}(\sigma)^T \otimes \mathrm{I}_f\right] \left(\mathrm{I}_f \otimes \tilde{A}(\sigma) + \tilde{A}(\sigma) \otimes \mathrm{I}_f\right)^{-1} \left(\tilde{A}(\sigma)^2 \otimes \tilde{A}(\sigma)^2\right) \\
&\times \mathrm{vec}\left(\frac{2}{\sigma^3} k_\sigma\left(WW^T\right) \odot \left(\mathbb{1}_{f\times f} - WW^T\right)\right) \\
&+ \left[P^T N(F) \otimes \tilde{A}(\sigma)\right] \\
&\times \mathrm{vec}\left(\frac{2}{\sigma^3} k_\sigma\left(WE(F)N(F)^{-1}\right) \odot \left(\mathbb{1}_{f\times p'} - WE(F)N(F)^{-1}\right)\right).
\end{aligned}
$$

□

A.5 Computation of the Smoothness Constants of a CKN

We present here the technical derivations and the mathematical proofs allowing us to obtain (1) an upper bound on the norm of the output of a CKN; (2) an upper bound on the Lipschitz constant of a CKN viewed as a function of its parameters; and (3) an upper bound on the Lipschitz constant of the gradient of a CKN viewed as a function of its parameters.

We now describe the main notations used throughout this section. We will use M to denote an upper bound on the norm of the output of a function, α to denote an upper bound on the Lipschitz constant of a function, and β to denote an upper bound on the Lipschitz constant of the gradient of a function. When subscripted solely by ℓ these values are the values corresponding to the ℓth layer of a CKN. Otherwise they indicate an intermediate quantity that has already been defined or is a constant in a definition. We will use the notation $\|\cdot\|$ to denote the Euclidean norm of vectors. The constants will be computed with respect to the Frobenius norm for simplicity.

A.5.1 Upper bounds

We begin with an upper bound on the *norm of the output of a CKN*; the proof is provided in Appendix A.5.4.

Proposition 37. *Consider a CKN with L layers defined in* (2.3)*. Recall that f_ℓ, p_ℓ, and s_ℓ are the number of filters, the total size of the patches, and the number of patches prior to pooling, respectively, at layer ℓ. Furthermore, recall that $\epsilon \mathrm{I}_{f_\ell}$ is the regularization added to $k(W_\ell W_\ell^T)$ at layer ℓ. Now define M_k and α_k to be upper bounds on the output of the kernel k and the Lipschitz constant of the kernel k, respectively. Finally, assume that when each patch is normalized the constant $\epsilon' \geq 0$ is added to its norm. An upper bound on the norm of the output of the CKN is given by M_L, defined through the recursion*

$$M_0 = \|F_0\|_F$$
$$M_\ell = \sqrt{\frac{f_\ell}{\epsilon}} \|P_\ell\|_F \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell} M_k\}(s_\ell M_{\ell-1}^2 + 2\sqrt{p_\ell} M_{\ell-1}\epsilon' + p_\ell \epsilon'^2)^{1/2}$$

for $\ell = 1, \ldots, L$.

Remark 38. *Considering a CKN with the Gaussian RBF kernel with bandwidth $\sigma \leq 1$, we can get the simplified expression*

$$M_\ell \leq 2\epsilon^{-1/2}\sqrt{p_\ell s_\ell} f_\ell \|P_\ell\|_F M_{\ell-1}$$

for $\ell = 1, \ldots, L$.

The upper bound scales as $\|P_\ell\|_F M_{\ell-1}$, if we set aside the term related to ϵ, the dimension-free terms, and the technical artifacts. For an architecture with a cuboid shape, that is, an architecture with layers having all the same properties, the upper bound grows exponentially as the network depth grows, driven by both the pooling-related terms $\|P_\ell\|_F$ and the dimension-related terms $f_\ell, \sqrt{p_\ell}$, and $\sqrt{s_\ell}$. This sheds light on architecture design. Indeed, for an architecture with a pyramidal shape, the growth of the upper bound as the network depth grows

can be mitigated by taking smaller and smaller patch sizes and numbers of filters as the layers are overlaid.

The general upper bound can be used to obtain an upper bound on the *Lipschitz constant of a CKN layer* viewed as a function of its parameters; the proof is provided in Section A.5.4.

Proposition 39. *Consider a CKN with L layers defined in* (2.3)*. An upper bound on the Lipschitz constant of the CKN is given by α_L, defined through the recursion*

$$
\begin{aligned}
&\alpha_0 = 1 \\
&\alpha_\ell = \sqrt{\frac{f_\ell s_\ell}{\epsilon}} \|P_\ell\|_F \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell} M_k\} \\
&\quad + \sqrt{\frac{2f_\ell}{\epsilon}} \|P_\ell\|_F \alpha_k (s_\ell M_{\ell-1}^2 + 2\sqrt{p_\ell} M_{\ell-1}\epsilon' + p_\ell \epsilon'^2)^{1/2} \max\{\sqrt{f_\ell}, \sqrt{p_\ell}\} \max\{1, 2\alpha_{\ell-1}/\epsilon'\} \\
&\quad + 2f_\ell \|P_\ell\|_F \alpha_k \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell} M_k\} (s_\ell M_{\ell-1}^2 + 2\sqrt{p_\ell} M_{\ell-1}\epsilon' + p_\ell \epsilon'^2)^{1/2} \\
&\qquad \times \left(\frac{f_\ell^{5/2}}{\epsilon^2} + \frac{f_\ell^{9/2}}{4\epsilon^{5/2}} \right)
\end{aligned}
$$

for $\ell = 1, \dots, L$.

Remark 40. *Consider again a CKN with the Gaussian RBF kernel. Assuming the bandwidth $\sigma \leq 1$, a simplified expression reads*

$$
\alpha_\ell \leq 12(\epsilon^{5/2}\epsilon')^{-1} \sqrt{s_\ell p_\ell} f_\ell^6 \alpha_k \|P_\ell\|_F M_{\ell-1} \alpha_{\ell-1}
$$

for $\ell = 1, \dots, L$.

The upper bound scales as $\|P_\ell\|_F M_{\ell-1}\alpha_{\ell-1}$, if we set aside the term related to ϵ, the dimension-free terms, and the technical artifacts. A similar reasoning can be applied, arguing in favor of architecture following a bottleneck in order to mitigate the exponential growth of the Lipschitz constant of the CKN.

Finally, we have the following proposition, which bounds the *Lipschitz constant of the gradient of a CKN*. The proof follows from the intermediate upper bounds given in Sections A.5.2 and A.5.3.

Proposition 41. *Consider a CKN with L layers defined in* (2.3)*. An upper bound on the Lipschitz constant of the gradient of the CKN is given by β_L, defined through the recursion*

$$\begin{aligned}\beta_0 &= 0 \\ \beta_\ell &= \alpha_{\ell-1}\beta_\ell^F + \alpha_\ell^F \beta_{\ell-1} + \mathcal{E}_\ell \beta_\ell^W \ , \qquad \ell = 1, \dots, L \ ,\end{aligned}$$

where α_ℓ is defined in Proposition 39, α_ℓ^F, β_ℓ^F, and β_ℓ^W are defined in Appendix A.5.3, $W = (\mathrm{vec}(W_1), \dots, \mathrm{vec}(W_L))$, and $\mathcal{E}_\ell \coloneqq \nabla_W \mathrm{vec}(W_\ell)$ is a matrix with ones in the entries for which the row and column correspond to the same element of W_ℓ and zeros elsewhere.

A.5.2 Lipschitz Constants of CKN Layer Components

Recall that a CKN layer may be written as

$$F_\ell = \Psi_\ell(F_{\ell-1}, W_\ell) \coloneqq \left(k(W_\ell W_\ell^T) + \epsilon \, \mathrm{I}\right)^{-1/2} k\left(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1}\right) N_\ell(F_{\ell-1}) P_\ell \ , \quad \text{(A.11)}$$

where $N_\ell(F_{\ell-1}) = N_\ell(E_\ell(F_{\ell-1}))$. Also recall that the filters in W_ℓ are constrained to have norm 1. In this subsection we compute upper bounds on the norms of the outputs and the Lipschitz constants of the components of a CKN layer ℓ.

Outline and strategy. The function implemented by a CKN layer given in Equation (A.11) involves functions acting on matrices, notably the functions $N_\ell(E_\ell(F_{\ell-1}))$ (and its inverse), $k(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1})$, and $(k(W_\ell W_\ell^T) + \epsilon \, \mathrm{I})^{-1/2}$. We first focus on the terms related to patch extraction and normalization. Next, we address the terms related to the action of the kernel. Afterward, we concentrate on the terms related to whitening in feature space involving the matrix square root and its inverse. Finally, we piece the terms together to

get the final estimates. Whenever derivations are similar, we shall focus on one example for clarity of exposition and omit the details of the other examples.

Patch extraction and normalization

First we prove bounds on the norms of the output of the patch extraction and normalization functions, along with bounds on the Lipschitz constants of these functions.

Lemma 42. *Define* $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{s_\ell \times p}$ *by*

$$f(A) = E_\ell(A) \ .$$

On the set $\mathcal{C} \coloneqq \{A \in \mathbb{R}^{m\times n} : \|A\| \leq M_A\}$ *for some* $M_A \geq 0$ *an upper bound on the norm of the output of* f *is*

$$M = \sqrt{s_\ell} M_A$$

and an upper bound on the Lipschitz constant of f *is*

$$\alpha = \sqrt{s_\ell} \ .$$

Proof. Note that for patches of total size s_ℓ each entry of A will appear in at most s_ℓ patches. Therefore, for all $A \in \mathcal{C}$,

$$\begin{aligned} \|f(A)\|_F^2 &\leq s_\ell \|A\|_F^2 \\ &\leq s_\ell M_A^2 \ . \end{aligned}$$

For the upper bound on the Lipschitz constant, observe that since patch extraction is a linear operator, we have that for all $A, B \in \mathcal{C}$

$$\|f(A) - f(B)\|_F = \|E_\ell(A) - E_\ell(B)\|_F$$

$$
\begin{aligned}
&= \|E_\ell(A - B)\|_F \\
&\leq \sqrt{s_\ell}\|A - B\|_F \ .
\end{aligned}
$$

□

Lemma 43. *Let $\epsilon \geq 0$ and define $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{p\times p}$ by*

$$f(A) = \operatorname{diag}([\|E_\ell(A)_{\cdot,1}\| + \epsilon, \ldots, \|E_\ell(A)_{\cdot,p}\| + \epsilon]) \ .$$

On the set $\mathcal{C} \coloneqq \{A \in \mathbb{R}^{m\times n} : \|A\| \leq M_A\}$ for some $M_A \geq 0$ an upper bound on the norm of the output of f is

$$M = (s_\ell M_A^2 + 2\sqrt{p}M_A\epsilon + p\epsilon^2)^{1/2}$$

and an upper bound on the Lipschitz constant of f is

$$\alpha = \sqrt{s_\ell} \ .$$

Proof. For the bound on the norm of the output of f, observe that for all $A \in \mathcal{C}$,

$$
\begin{aligned}
\|f(A)\|_F^2 &= (\|E_\ell(A)_{\cdot,1}\| + \epsilon)^2 + \cdots + (\|E_\ell(A)_{\cdot,p}\| + \epsilon)^2 \\
&= \|E_\ell(A)\|_F^2 + 2\epsilon \sum_{i=1}^{p} \|E_\ell(A)_{\cdot,i}\|_F + p\epsilon^2 \\
&\leq s_\ell M_A^2 + 2\sqrt{p}\epsilon\|E_\ell(A)\|_F + p\epsilon^2 \\
&\leq s_\ell M_A^2 + 2\sqrt{p}M_A\epsilon + p\epsilon^2 \ ,
\end{aligned}
$$

where the second-to-last line follows from Hölder's inequality. For the upper bound on the Lipschitz constant of f, observe that for all $A, B \in \mathcal{C}$ we have

$$\|f(A) - f(B)\|_F^2 = \|\operatorname{diag}([\|E_\ell(A)_{\cdot,1}\| - \|E_\ell(B)_{\cdot,1}\|, \ldots, \|E_\ell(A)_{\cdot,p}\| - \|E_\ell(B)_{\cdot,p}\|])\|_F^2$$

$$
\begin{aligned}
&= \sum_{i=1}^{p} (\|E_\ell(A)_{\cdot,i}\| - \|E_\ell(B)_{\cdot,i}\|)^2 \\
&\leq \sum_{i=1}^{p} \|E_\ell(A)_{\cdot,i} - E_\ell(B)_{\cdot,i}\|^2 \\
&= \|E_\ell(A) - E_\ell(B)\|_F^2 \\
&\leq s_\ell \|A - B\|_F^2 \, .
\end{aligned}
$$

□

Lemma 44. *Let $\epsilon > 0$ and define $f : \mathbb{R}^d \to \mathbb{R}^d$ by*

$$f(x) = \frac{x}{\|x\| + \epsilon} \, .$$

An upper bound on the norm of the output of f is

$$M = 1 \, ,$$

an upper bound on the Lipschitz constant of f is

$$\alpha = \frac{2}{\epsilon} \, ,$$

and an upper bound on the Lipschitz constant of ∇f is

$$\beta = \frac{7 + \sqrt{d}}{\epsilon^2} \, .$$

Proof. The first result is trivial. Now we will compute an upper bound on the Lipschitz constant of f. Observe that for all $x, y \in \mathbb{R}^d$

$$\left\| \frac{x}{\|x\| + \epsilon} - \frac{y}{\|y\| + \epsilon} \right\| \leq \frac{2}{\epsilon} \|x - y\| \, .$$

Next we will compute an upper bound on the Lipschitz constant of ∇f. Note that

$$\nabla f(x) = \begin{cases} \underbrace{\frac{1}{\|x\|+\epsilon}\,\mathrm{I}_d}_{=:g(x)} - \underbrace{\frac{1}{(\|x\|+\epsilon)^2}\left(\frac{1}{\|x\|}\right)xx^T}_{=:h(x)}\,, & x \neq 0 \\ \frac{1}{\|x\|+\epsilon}\,\mathrm{I}_d, & x = 0\,. \end{cases}$$

Without loss of generality, assume that $\|x\| \geq \|y\|$. First, observe that

$$\left\| \frac{1}{\|x\|+\epsilon}\,\mathrm{I}_d - \frac{1}{\|y\|+\epsilon}\,\mathrm{I}_d \right\|_F \leq \frac{\sqrt{d}}{\epsilon^2}\|x-y\|\,.$$

Hence, an upper bound on the Lipschitz constant of g is $\sqrt{d}/\epsilon^2$. Next, note that for $\|x\|, \|y\| \neq 0$,

$$\begin{aligned} &\left\| \frac{xx^T}{\|x\|(\|x\|+\epsilon)^2} - \frac{yy^T}{\|y\|(\|y\|+\epsilon)^2} \right\|_F \\ &\leq \frac{1}{\|x\|(\|x\|+\epsilon)^2}\left\|xx^T - yy^T\right\|_F + \|y\|^2\left\| \frac{1}{\|x\|(\|x\|+\epsilon)^2} - \frac{1}{\|y\|(\|y\|+\epsilon)^2} \right\|. \end{aligned}$$

Using the identity

$$(x-y)(x+y)^T - (x-y)y^T - y(y-x)^T = xx^T - yy^T\,,$$

we have

$$\begin{aligned} &\left\| \frac{xx^T}{\|x\|(\|x\|+\epsilon)^2} - \frac{yy^T}{\|y\|(\|y\|+\epsilon)^2} \right\|_F \\ &\leq \frac{7}{\epsilon^2}\|x-y\|\,. \end{aligned}$$

In the case where $\|y\| = 0$, we have

$$\left\| \frac{xx^T}{(\|x\|+\epsilon)^2\|x\|} - 0 \right\|_F \leq \frac{1}{(\|x\|+\epsilon)^2}\|x\| \leq \frac{1}{\epsilon^2}\|x\|\,.$$

Hence, an upper bound on the Lipschitz constant of h is $7/\epsilon^2$. The overall bound on the Lipschitz constant of ∇f is thus $(7 + \sqrt{d})/\epsilon^2$. □

Lemma 45. *Let $\epsilon > 0$ and define $f : \mathbb{R}^d \to \mathbb{R}^d$ by*

$$f(x) = \frac{x}{\|x\| + \epsilon}$$

and $g : \mathbb{R}^{d\times p} \to \mathbb{R}^{d\times p}$ by

$$g(A) = [f(A_{\cdot,1}), \ldots, f(A_{\cdot,p})] \,.$$

An upper bound on the norm of the output of g is

$$M = \sqrt{p} \,,$$

an upper bound on the Lipschitz constant of g is

$$\alpha = \frac{2}{\epsilon} \,,$$

and an upper bound on the Lipschitz constant of ∇g is

$$\beta = \frac{7 + \sqrt{d}}{\epsilon^2} \,.$$

Proof. The proofs follow from Lemma 44. □

Nonlinearities

We consider four different dot product kernels: the arc-cosine kernels of order 0 and 1, the RBF kernel on the sphere, and the Matérn kernel on the sphere. Next we compute upper bounds on the Lipschitz constant of each dot-product kernel and its derivative, if they exist.

Lemma 46 (Arc-cosine kernel of order 0)**.** *Define* $f : [-1, 1] \to \mathbb{R}$ *by*

$$f(x) = 1 - \frac{1}{\pi}\cos^{-1}(x) \ .$$

The range of f *is* $[0, 1]$*. The function* f *is not Lipschitz. Moreover,* f *is not differentiable.*

Proof. The range of the arc-cosine function is $[0, \pi]$. Therefore, $f(x) \in [0, 1]$. Now let $x = 1$. Then $f(x) = 1$ and we have

$$\begin{aligned}\lim_{y\to 1^-} \frac{\|f(x) - f(y)\|}{\|x - y\|} &= \lim_{y\to 1^-} \frac{\cos^{-1}(y)}{\pi(1-y)} \\ &= \lim_{y\to 1^-} \frac{1}{\pi\sqrt{1-y^2}} \\ &= \infty \ .\end{aligned}$$

Therefore, there does not exist a value α such that $\|f(x) - f(y)\| \leq \alpha\|x - y\|$ for all $x, y \in [-1, 1]$ and so f is not Lipschitz.

Next, note that for $x \in (-1, 1)$,

$$f'(x) = \frac{1}{\pi\sqrt{1-x^2}} \ .$$

As the limit of the above function is not finite at ± 1 we cannot define a derivative at ± 1 and so f is not differentiable. □

Lemma 47 (Arc-cosine kernel of order 1)**.** *Define* $f : [-1, 1] \to \mathbb{R}$ *by*

$$f(x) = \frac{1}{\pi}\left[\sin(\cos^{-1}(x)) + (\pi - \cos^{-1}(x))x\right] \ .$$

The range of f *is* $[0, 1]$*. The Lipschitz constant of* f *is*

$$\alpha = 1 \ .$$

The derivative of f is not Lipschitz.

Proof. Note that

$$f(x) = \frac{1}{\pi}\left[\sqrt{1-x^2} + (\pi - \cos^{-1}(x))x\right] .$$

For $x \in (-1, 1)$ we have

$$\begin{aligned} f'(x) &= \frac{1}{\pi}\left[\frac{-x}{\sqrt{1-x^2}} + \pi - \cos^{-1}(x) + \frac{x}{\sqrt{1-x^2}}\right] \\ &= 1 - \frac{1}{\pi}\cos^{-1}(x) \end{aligned}$$

and $\lim_{x\to -1^+} 1 - \cos^{-1}(x)/\pi = 0$ and $\lim_{x\to 1^-} 1 - \cos^{-1}(x)/\pi = 1$. Therefore, the Lipschitz constant of f is $\alpha = 1$. Since f' is positive, the range of f is $[f(-1), f(1)] = [0, 1]$. By Lemma 46 the derivative of f is not Lipschitz. □

Lemma 48 (Gaussian RBF kernel on the sphere)**.** *Let $\sigma > 0$. Define $f : [-1, 1] \to \mathbb{R}$ by*

$$f(x; \sigma) = \exp\left(\frac{x-1}{\sigma^2}\right) .$$

The range of f is $[\exp(-2/\sigma^2), 1]$. The Lipschitz constant of f is

$$\alpha = \frac{1}{\sigma^2}$$

and the Lipschitz constant of the derivative of f is

$$\beta = \frac{1}{\sigma^4} .$$

Proof. The derivative of f is

$$f'(x; \sigma) = \frac{1}{\sigma^2}\exp\left(\frac{x-1}{\sigma^2}\right) .$$

Therefore,

$$|f'(x;\sigma)| \leq \frac{1}{\sigma^2} ,$$

with equality when $x = 1$. The second derivative of f is

$$f''(x;\sigma) = \frac{1}{\sigma^4} \exp\left(\frac{x-1}{\sigma^2}\right) .$$

Therefore,

$$|f''(x;\sigma)| \leq \frac{1}{\sigma^4} ,$$

with equality when $x = 1$. □

Lemma 49 (Matérn kernel on the sphere)**.** *Let $\sigma > 0$ and $\nu > 0$. Define $f : [-1, 1] \to \mathbb{R}$ by*

$$f(x;\sigma,\nu) = G_\nu\left(\frac{2\sqrt{\nu(1-x)}}{\sigma}\right) ,$$

where

$$G_\nu(z) \coloneqq \frac{1}{2^{\nu-1}\Gamma(\nu)} z^\nu K_\nu(z) ,$$

K_ν is the modified Bessel function of the second kind of order ν, and Γ is the Gamma function. The range of f is $[G_\nu(2\sqrt{2\nu}/\sigma), 1]$. The function f is Lipschitz if and only if $\nu > 1$, with constant

$$\alpha = \frac{\nu}{\sigma^2(\nu-1)}$$

and the derivative of f is Lipschitz if and only if $\nu > 2$, with constant

$$\beta = \frac{\nu^2}{\sigma^4(\nu-1)(\nu-2)} .$$

Proof. Assume $\nu > 1$. By Lemma 62 in Appendix A.8, the derivative of f is given by

$$f'(x;\sigma,\nu) = \frac{1}{\sigma'^2} f(x;\sigma',\nu-1) \ ,$$

with $\sigma' = \sigma\sqrt{\nu-1}/\sqrt{\nu}$. Note that since f is always positive, $f' > 0$ for all x. At $x = 1$, f reaches its maximum of 1. Therefore,

$$\begin{aligned} |f'(x;\sigma,\nu)| &\leq \frac{1}{\sigma'^2} \\ &= \frac{\nu}{\sigma^2(\nu-1)} \ , \end{aligned}$$

with equality when $x = 1$. Moreover, as noted in the proof of Lemma 62, for $0 < \nu \leq 1$ the derivative diverges as $x \to 1$. Therefore, f is Lipschitz if and only if $\nu > 1$, with constant $\nu/(\sigma^2(\nu-1))$.

Next, assume $\nu > 2$. Differentiating again, we find that

$$f''(x;\sigma,\nu) = \frac{1}{\sigma'^2\sigma''^2} f(x;\sigma'',\nu-2) \ ,$$

with $\sigma'' = \sigma'\sqrt{\nu-2}/\sqrt{\nu-1}$. Therefore,

$$\begin{aligned} |f''(x;\sigma,\nu)| &\leq \frac{1}{\sigma'^2\sigma''^2} \\ &= \frac{\nu-1}{\sigma'^4(\nu-2)} \\ &= \frac{\nu^2(\nu-1)}{\sigma^4(\nu-1)^2(\nu-2)} \\ &= \frac{\nu^2}{\sigma^4(\nu-1)(\nu-2)} \ , \end{aligned}$$

with equality when $x = 1$. Therefore, f' is Lipschitz if and only if $\nu > 2$, with constant $\nu^2/(\sigma^4(\nu-1)(\nu-2))$.

□

Matrix inverse square root

Now we focus on the matrix inverse square root term.

Lemma 50. *Fix $\epsilon > 0$. Define $f : \mathbb{S}^n_+ \to \mathbb{S}^n_{++}$ by*

$$f(A) = (A + \epsilon \, \mathrm{I}_n)^{-1/2} \, .$$

An upper bound on the norm of the output of f is

$$M = \sqrt{\frac{n}{\epsilon}} \, ,$$

an upper bound on the Lipschitz constant of f is

$$\alpha = \frac{n^2}{2\epsilon^{3/2}} \, ,$$

and an upper bound on the Lipschitz constant of ∇f is

$$\beta = \frac{n^{5/2}}{\epsilon^2} + \frac{n^{9/2}}{4\epsilon^{5/2}} \, .$$

Proof. For the upper bound on the norm of the output of f, note that

$$\begin{aligned} \|f(A)\|_F &\leq \sqrt{n} \|(A + \epsilon \, \mathrm{I}_n)^{-1/2}\|_2 \\ &= \frac{\sqrt{n}}{\lambda_{\min}((A + \epsilon \, \mathrm{I}_n)^{1/2})} \\ &\leq \sqrt{\frac{n}{\epsilon}} \, . \end{aligned}$$

Now define $e(A) = A + \epsilon \, \mathrm{I}$, $g(A) = A^{1/2}$, and $h(A) = A^{-1}$. Then $h(g(e(A))) = (A + \epsilon \, \mathrm{I})^{-1/2}$ and $\nabla_{\mathrm{vec}(A)} \, \mathrm{vec}(h(g(e(A)))) = \nabla_{\mathrm{vec}(B)} \, \mathrm{vec}(g(B))|_{e(A)} \nabla_{\mathrm{vec}(C)} \, \mathrm{vec}(h(C))|_{g(e(A))}$. The upper bound on the Lipschitz constant of f follows from bounds on the Lipschitz constant of the composition of functions, the inverse of a matrix, and the square root of a matrix and is

given by

$$\alpha = \left(\frac{n}{2\sqrt{\epsilon}}\right)\left(\frac{n}{\epsilon}\right) = \frac{n^2}{2\epsilon^{3/2}} \ .$$

We now focus on the upper bound on the Lipschitz constant of ∇f. Using again bounds on the Lipschitz constant of the inverse and square root of a matrix, in addition to a bound on the Lipschitz constant of a product of bounded matrices, we have that an upper bound on the Lipschitz constant of ∇f is

$$\begin{aligned}\beta &= \left(\frac{n}{2\sqrt{\epsilon}}\right)\left(\frac{2n^{3/2}}{\epsilon^{3/2}}\right) + \left(\frac{n}{\epsilon}\right)\left(\frac{n^{7/2}}{4\epsilon^{3/2}}\right) \\ &= \frac{n^{5/2}}{\epsilon^2} + \frac{n^{9/2}}{4\epsilon^{5/2}} \ .\end{aligned}$$

□

Larger CKN layer components

Using the results above we now compute upper bounds on the norms of the outputs and the Lipschitz constants of the larger components of a CKN layer.

Lemma 51. *Let $\epsilon > 0$ and define $f : \mathbb{R}^{m\times n} \to \mathbb{S}^m_{++}$ by*

$$f(A) = \left(k(AA^T) + \epsilon\,\mathrm{I}\right)^{-1/2} \ ,$$

where $k : \mathbb{R} \to \mathbb{R}$ is understood to be applied element-wise, its outputs are bounded by $M_k \leq \infty$, and it is Lipschitz with constant α_k. An upper bound on the norm of the output of f is

$$M = \sqrt{\frac{m}{\epsilon}}$$

and on the set $\mathcal{C} \coloneqq \{A \in \mathbb{R}^{m\times n} : \|A\| \leq M_A\}$ for some $M_A \geq 0$ an upper bound on the

Lipschitz constant of f is given by

$$\alpha = 2M_A\alpha_k\left(\frac{m^{5/2}}{\epsilon^2} + \frac{m^{9/2}}{4\epsilon^{5/2}}\right) .$$

In particular, for $A = W_\ell$ we have $M = \sqrt{f_\ell/\epsilon}$ and $\alpha = 2f_\ell\alpha_k\left(\frac{f_\ell^{5/2}}{\epsilon^2} + \frac{f_\ell^{9/2}}{4\epsilon^{5/2}}\right)$.

Proof. The bound on the norm of the output of f follows from Lemma 50. Using Lemma 50, along with bounds on the Lipschitz constants of the composition of functions and product of bounded matrices, we have that the Lipschitz constant is bounded above by

$$\alpha = 2M_A\alpha_k\left(\frac{m^{5/2}}{\epsilon^2} + \frac{m^{9/2}}{4\epsilon^{5/2}}\right) .$$

□

Lemma 52. *Define $f : \mathbb{R}^{m\times n} \times \mathbb{R}^{p\times q} \to \mathbb{R}^{m\times r}$ by*

$$f(A, B) = k\left(AE_\ell(B)N_\ell(B)^{-1}\right) ,$$

where $k : \mathbb{R} \to \mathbb{R}$ is understood to be applied element-wise, its outputs are bounded by $M_k \leq \infty$ and it is Lipschitz with constant α_k. On the set $\mathcal{C} \coloneqq \{A \in \mathbb{R}^{m\times n} : \|A\|_F \leq M_A\} \times \mathbb{R}^{p\times q}$ for some $M_A \geq 0$ an upper bound on the norm of the output of f is

$$M = \min\{\sqrt{r}M_A\alpha_k + \sqrt{mr}|k(0)|, \sqrt{mr}M_k\}$$

and an upper bound on the Lipschitz constant of f is

$$\alpha = \sqrt{2}\max\{M_A, \sqrt{p_\ell}\}\max\left\{1, \frac{2}{\epsilon}\right\}\alpha_k .$$

In particular, for $A = W_\ell$ and $B = F_{\ell-1}$ we have $M = \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell}M_k\}$ and $\alpha = \sqrt{2}\max\{\sqrt{f_\ell}, \sqrt{p_\ell}\}\max\{1, 2/\epsilon\}\alpha_k$.

Proof. We can bound the norm of the output of f in two ways. First, we can use the Lipschitz continuity of k to obtain

$$\begin{aligned}\|k\left(AE_\ell(B)N_\ell(B)^{-1}\right)\|_F - \|k(0_{m\times r})\|_F &\leq \|k\left(AE_\ell(B)N_\ell(B)^{-1}\right) - k\left(0_{m\times r}\right)\|_F \\ &\leq \alpha_k \|AE_\ell(B)N_\ell(B)^{-1}\|_F \\ &\leq \alpha_k \|A\|_F \|E_\ell(B)N_\ell(B)^{-1}\|_F \\ &\leq \sqrt{r} M_A \alpha_k \ .\end{aligned}$$

Alternatively, we can use the bound

$$\|k\left(AE_\ell(B)N_\ell(B)^{-1}\right)\|_F \leq \sqrt{mr} M_k \ .$$

Therefore, the norm of the output of f is bounded by

$$M = \min\{\sqrt{r} M_A \alpha_k + \sqrt{mr}|k(0)|, \sqrt{mr} M_k\} \ .$$

An upper bound on the Lipschitz constant follows from Lemma 45 and upper bounds on the Lipschitz constants of the composition of functions and product of bounded matrices, and is given by

$$\alpha = \sqrt{2}\max\{M_A, \sqrt{p_\ell}\} \max\left\{1, \frac{2}{\epsilon}\right\} \alpha_k \ .$$

□

Based on the above lemmas we define the following quantities, which will be used in bounding the norm of the output of a CKN layer and in bounding the Lipschitz constant of a CKN layer:

- $M_{A_\ell} = \sqrt{f_\ell/\epsilon}$

- $M_{B_\ell} = \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell}M_k\}$
- $M_{C_\ell} = (s_\ell M_{\ell-1}^2 + 2\sqrt{p_\ell}M_{\ell-1}\epsilon' + p_\ell\epsilon'^2)^{1/2}\|P_\ell\|_F$
- $\alpha_{A_\ell} = 2f_\ell\alpha_k\left(\frac{f_\ell^{5/2}}{\epsilon^2} + \frac{f_\ell^{9/2}}{4\epsilon^{5/2}}\right)$
- $\alpha_{B_\ell} = \sqrt{2}\max\{\sqrt{f}_\ell, \sqrt{p_\ell}\}\max\{1, 2\alpha_{\ell-1}/\epsilon'\}\alpha_k$
- $\alpha_{C_\ell} = \sqrt{s_\ell}\|P_\ell\|_F$

where we distinguish between the constant ϵ added to the matrix $k(W_\ell W_\ell^T)$ and the constant ϵ' added when normalizing patches. Here we have included the pooling term $\|P_\ell\|_F$ with the normalization term and have taken into consideration the fact that $F_{\ell-1} = \Psi(F_{\ell-2})$ and hence the Lipschitz constant of the term $k\left(W_\ell E_\ell(F_{\ell-1})N_\ell(F_{\ell-1})^{-1}\right)$ depends on the Lipschitz constant $\alpha_{\ell-1}$ of the previous CKN layer. We define $M_0 = \|F_0\|_F$ and $\alpha_0 = 1$.

A.5.3 Lipschitz Constants of the CKN Gradient Components

We upper bound the Lipschitz constant of the gradient of a CKN layer by computing the bounds for its components and then combining them using Lipschitz bounds on fundamental quantities.

The proofs follow arguments similar to the previous proofs. The upper bounds result from considering each term in turn in the following equalities. First, recall from Proposition 27 in Appendix A.4 that the terms from $\nabla_{\mathrm{vec}(W_\ell)}\mathrm{vec}(\Psi_\ell)$ expand as follows:

$$\begin{aligned}\nabla_{\mathrm{vec}(W_\ell)}\mathrm{vec}(\Psi_\ell) = & -\left[(B(W_\ell, F_{\ell-1})N_\ell(F_{\ell-1})P_\ell)^T \otimes \mathrm{I}_{f_\ell}\right] \\ & \times (\mathrm{I}_{f_\ell}\otimes A(W_\ell) + A(W_\ell)\otimes \mathrm{I}_{f_\ell})^{-1}\left(A(W_\ell)^2 \otimes A(W_\ell)^2\right) \\ & \times \mathrm{diag}\left[\mathrm{vec}\left(k'\left(W_\ell W_\ell^T\right)\right)\right](W_\ell \otimes \mathrm{I}_{f_\ell} + (\mathrm{I}_{f_\ell}\otimes W_\ell)T_{f_\ell,s_\ell}) \\ & + \left[(N_\ell(F_{\ell-1})P_\ell)^T \otimes A(W_\ell)\right] \\ & \times \mathrm{diag}\left[\mathrm{vec}\left(k'\left(W_\ell E_\ell(F_{\ell-1})N_\ell(F_{\ell-1})^{-1}\right)\right)\right]\end{aligned}$$

$$\times \left[(E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1})^T \otimes \mathrm{I}_{f_\ell} \right] \ .$$

Second, recall from Proposition 33 in Appendix A.4 that the terms from $\nabla_{\mathrm{vec}(F_{\ell-1})} \mathrm{vec}(\Psi_\ell)$ expand as follows:

$$\begin{aligned}
&\nabla_{\mathrm{vec}(F_{\ell-1})} \mathrm{vec}(\Psi_\ell) \\
=&\Bigg\{ \Bigg(\left[P_\ell^T N_\ell(F_{\ell-1}) \otimes A(W_\ell) \right] \mathrm{diag} \left[\mathrm{vec} \left(k'(W_\ell E_\ell(F_{\ell-1}) N_\ell(F_{\ell-1})^{-1} \right) \right] \\
&\quad \times \Big[\left[N_\ell(F_{\ell-1})^{-1} \otimes W_\ell \right] \\
&\qquad - \left[\mathrm{I}_{p_\ell} \otimes (W_\ell E_\ell(F_{\ell-1})) \right] \mathrm{diag} \left[\mathrm{vec} \left(N_\ell(F_{\ell-1})^{-3} \right) \right] \times \left[\mathrm{I}_{p_\ell} \otimes E_\ell(F_{\ell-1})^T \right] \Big] \Bigg) \\
&+ \left[P_\ell^T \otimes (A(W_\ell) B(W_\ell, F_{\ell-1})) \right] \mathrm{diag} \left[\mathrm{vec} \left(N_\ell(F_{\ell-1})^{-1} \right) \right] \times \left[\mathrm{I}_{p_\ell} \otimes E_\ell(F_{\ell-1})^T \right] \Bigg\} \times \Sigma_\ell \ ,
\end{aligned}$$

where $\Sigma_\ell = \sum_{i=1}^{s_\ell / f_{\ell-1}} \left(E_{\ell 2 i}^T \otimes E_{\ell 1 i} \right)$.

Now define the following quantities:

- $M_{D_\ell} = \sqrt{f_\ell} M_{B_\ell} M_{C_\ell}$
- $M_{E_\ell} = M_{A_\ell}^2$
- $M_{F_\ell} = \frac{f_\ell}{2\sqrt{\epsilon}}$
- $M_{G_\ell} = \min\{ f_\ell \alpha_k, f_\ell \beta_k + f_\ell |k'(0)| \}$
- $M_{H_\ell} = (1 + \sqrt{f_\ell s_\ell}) f_\ell$
- $M_{I_\ell} = M_{A_\ell} M_{C_\ell}$
- $M_{J_\ell} = \min\{ \sqrt{f_\ell p_\ell} \beta_k + \sqrt{f_\ell p_\ell} |k'(0)|, \sqrt{f_\ell p_\ell} \alpha_k \}$

- $M_{K_\ell} = \sqrt{f_\ell p_\ell}$
- $M_{L_\ell} = \frac{\sqrt{f_\ell p_\ell}}{\epsilon}$
- $M_{M_\ell} = \sqrt{f_\ell p_\ell s_\ell} M_{\ell-1}$
- $M_{N_\ell} = \sqrt{p_\ell s_\ell} M_{\ell-1}$
- $M_{O_\ell} = \frac{\sqrt{p_\ell}}{\epsilon}$
- $M_{P_\ell} = \frac{\sqrt{p_\ell}}{\epsilon^3}$
- $M_{Q_\ell} = M_{A_\ell} M_{B_\ell} \|P_\ell\|_F$
- $\alpha'_{B_\ell} = \sqrt{2} \max\{\sqrt{f_\ell}, \sqrt{p_\ell}\} \max\{1, 2/\epsilon'\} \alpha_k$
- $\alpha_{D_\ell} = \sqrt{f_\ell}(M_{B_\ell} \alpha_{C_\ell} + M_{C_\ell} \alpha'_{B_\ell})$
- $\alpha_{E_\ell} = 2 M_{A_\ell} \alpha_{A_\ell}$
- $\alpha_{F_\ell} = \frac{f_\ell^4 \alpha_k}{2\epsilon^{3/2}}$
- $\alpha_{G_\ell} = 2\sqrt{f_\ell} \beta_k$
- $\alpha_{H_\ell} = (1 + \sqrt{f_\ell s_\ell})\sqrt{f_\ell}$
- $\alpha_{I_\ell} = \sqrt{2} \max\{M_{A_\ell}, M_{C_\ell}\} \max\{\alpha_{A_\ell}, \alpha_{C_\ell}\}$
- $\alpha_{J_\ell} = \sqrt{2} \max\{\sqrt{f}_\ell, \sqrt{p_\ell}\} \max\left\{1, \frac{2}{\epsilon}\right\} \beta_k$
- $\alpha_{K_\ell} = \frac{2\sqrt{f_\ell}}{\epsilon}$
- $\alpha_{L_\ell} = \sqrt{2} \max\{\sqrt{\frac{p_\ell}{\epsilon}}, \sqrt{f_\ell}\} \max\left\{\frac{\sqrt{s_\ell}}{\epsilon^2}, 1\right\}$

- $\alpha_{M_\ell} = \sqrt{2p_\ell} \max\{\sqrt{f}_\ell, \sqrt{s}_\ell M_{\ell-1}\} \max\{1, \sqrt{s_\ell}\}$
- $\alpha_{N_\ell} = \sqrt{p_\ell s_\ell}$
- $\alpha_{O_\ell} = \frac{\sqrt{s_\ell}}{\epsilon^2}$
- $\alpha_{P_\ell} = 3\frac{p_\ell \sqrt{s_\ell}}{\epsilon^4}$
- $\alpha_{Q_\ell} = (M_{A_\ell}\alpha'_{B_\ell} + M_{B_\ell}\alpha_{A_\ell})\|P_\ell\|_F$.

We then have the following bounds on components of the CKN gradient.

Lemma 53. *An upper bound on the norm of the term* $\nabla_{\mathrm{vec}(F_{\ell-1})} \mathrm{vec}(\Psi_\ell(F_{\ell-1}, W_\ell))$ *at each layer* ℓ *of a CKN is given by* α^F_ℓ*, defined as*

$$\alpha^F_0 = 0$$
$$\alpha^F_\ell = \{M_{I_\ell} M_{J_\ell}(M_{L_\ell} + M_{M_\ell} M_{N_\ell} M_{P_\ell}) + M_{N_\ell} M_{O_\ell} M_{Q_\ell}\}\|\Sigma_\ell\|_F \ , \quad \ell = 1, \ldots, L \ .$$

Lemma 54. *An upper bound on the Lipschitz constant of the term* $\nabla_{\mathrm{vec}(W_\ell)} \mathrm{vec}(\Psi_\ell(F_{\ell-1}, W_\ell))$ *at each layer* ℓ *of a CKN is given by* β^W_ℓ*, defined as*

$$\begin{aligned}\beta^W_\ell = {} & M_{D_\ell} M_{E_\ell} M_{F_\ell} M_{G_\ell} \alpha_{H_\ell} + M_{D_\ell} M_{E_\ell} M_{F_\ell} \alpha_{G_\ell} M_{H_\ell} + M_{D_\ell} M_{E_\ell} \alpha_{F_\ell} M_{G_\ell} M_{H_\ell} \\ & + M_{D_\ell} \alpha_{E_\ell} M_{F_\ell} M_{G_\ell} M_{H_\ell} + \alpha_{D_\ell} M_{E_\ell} M_{F_\ell} M_{G_\ell} M_{H_\ell} + \\ & + M_{I_\ell} M_{J_\ell} \alpha_{K_\ell} + M_{I_\ell} \alpha_{J_\ell} M_{K_\ell} + \alpha_{I_\ell} M_{J_\ell} M_{K_\ell} \ , \quad \ell = 1, \ldots, L \ .\end{aligned}$$

Lemma 55. *An upper bound on the Lipschitz constant of the term* $\nabla_{\mathrm{vec}(F_{\ell-1})} \mathrm{vec}(\Psi_\ell(F_{\ell-1}, W_\ell))$ *at each layer* ℓ *of a CKN is given by* β^F_ℓ*, defined as*

$$\beta^F_0 = 0$$
$$\beta^F_\ell = \{M_{I_\ell} M_{J_\ell} \alpha_{LMNP_\ell} + M_{I_\ell} \alpha_{J_\ell} M_{LMNP_\ell} + \alpha_{I_\ell} M_{J_\ell} M_{LMNP_\ell} + \alpha_{NOQ_\ell}\} \|\Sigma_\ell\|_F \ , \ \ \ell = 1, \ldots, L \ ,$$

where

$$
\begin{aligned}
M_{LMNP_\ell} &= M_{L_\ell} + M_{M_\ell} M_{N_\ell} M_{P_\ell} \ ; \\
\alpha_{LMNP_\ell} &= \alpha_{L_\ell} + M_{M_\ell} M_{N_\ell} \alpha_{P_\ell} + M_{M_\ell} \alpha_{N_\ell} M_{P_\ell} + \alpha_{M_\ell} M_{N_\ell} M_{P_\ell} \ ; \ \textit{and} \\
\alpha_{NOQ_\ell} &= M_{N_\ell} M_{O_\ell} \alpha_{Q_\ell} + M_{N_\ell} \alpha_{O_\ell} M_{Q_\ell} + \alpha_{N_\ell} M_{O_\ell} M_{Q_\ell} \ .
\end{aligned}
$$

A.5.4 Proofs of the final upper bound estimates

Proof of Proposition 37. We have that the output at layer $\ell \geq 1$ is bounded by

$$
\begin{aligned}
M_\ell &= M_{A_\ell} M_{B_\ell} M_{C_\ell} \\
&= \sqrt{\frac{f_\ell}{\epsilon}} \|P_\ell\|_F \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell} M_k\} (s_\ell M_{\ell-1}^2 + 2\sqrt{p_\ell} M_{\ell-1}\epsilon' + p_\ell \epsilon'^2)^{1/2} \ .
\end{aligned}
$$

□

Proof of Proposition 39. We have that the Lipschitz constant of a CKN at layer ℓ is bounded by

$$
\begin{aligned}
\alpha_\ell &= M_{A_\ell} M_{B_\ell} \alpha_{C_\ell} + M_{A_\ell} M_{C_\ell} \alpha_{B_\ell} + M_{B_\ell} M_{C_\ell} \alpha_{A_\ell} \\
&= \sqrt{\frac{f_\ell s_\ell}{\epsilon}} \|P_\ell\|_F \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell} M_k\} \\
&\quad + \sqrt{\frac{2f_\ell}{\epsilon}} \|P_\ell\|_F \alpha_k (s_\ell M_{\ell-1}^2 + 2\sqrt{p_\ell} M_{\ell-1}\epsilon' + p_\ell \epsilon'^2)^{1/2} \max\{\sqrt{f}_\ell, \sqrt{p_\ell}\} \max\{1, 2\alpha_{\ell-1}/\epsilon'\} \\
&\quad + 2f_\ell \|P_\ell\|_F \alpha_k \min\{\sqrt{f_\ell p_\ell}\alpha_k + \sqrt{f_\ell p_\ell}|k(0)|, \sqrt{f_\ell p_\ell} M_k\} (s_\ell M_{\ell-1}^2 + 2\sqrt{p_\ell} M_{\ell-1}\epsilon' + p_\ell \epsilon'^2)^{1/2} \\
&\qquad \times \left(\frac{f_\ell^{5/2}}{\epsilon^2} + \frac{f_\ell^{9/2}}{4\epsilon^{5/2}} \right) \ .
\end{aligned}
$$

□

A.6 Stochastic Gradient Optimization on a Product of Spheres

In this section we detail the plain stochastic gradient optimization on a product of spheres algorithm used to train a CKN and establish its convergence to a stationary point.

For better readability, denote by $w_\ell = \mathrm{Vect}(W_\ell) \in \mathbb{R}^{d_\ell}$ the vectorized weights at the ℓth layer with $d_\ell = f_\ell \times s_\ell$ parameters and $w_{1:L+1} = (w_1; \dots; w_{L+1})$ the concatenation of those layers. Furthermore, denote by $S^{d_\ell} = \prod_{j=1}^{f_\ell} \mathbb{S}^{s_\ell}$ the Cartesian product of Euclidean unit spheres in $\mathbb{R}^{s_\ell}$ and by $\mathcal{B}_{2,\lambda}$ the Euclidean ball centered at the origin of radius λ. The problem then reads

$$\begin{aligned} \min_{w_{1:L+1}} \quad & f(w_{1:L+1}) \coloneqq \frac{1}{n} \sum_{i=1}^{n} f_i(w_{1:L+1}) \\ \text{subject to} \quad & w_{1:L+1} \in \mathcal{C} \coloneqq S^{d_0} \times \cdots \times S^{d_{L-1}} \times \mathcal{B}_{2,\lambda} \end{aligned} \tag{A.12}$$

where $f_i(w_{1:L+1})$ is the loss incurred on the ith sample and the set of constraints $\mathcal{C}$ is a product of manifolds and is therefore a manifold itself.

Convergence analysis for optimization on manifolds hinges upon smooth curves, called retractions, parametrized by a point on the manifold and a direction. In our case, they amount to block-coordinate projections on the sphere. Formally, with a given set of weights $w_{1:L+1}$, given a direction $\delta_{1:L+1} = (\delta_1; \dots; \delta_{L+1})$ where δ_ℓ denotes the portion corresponding to the ℓth layer, the retraction is defined as

$$\begin{aligned} & \mathrm{ret}(\delta_{1:L+1}; w_{1:L+1}) = v_{1:L+1} \\ \text{where} \quad & v_\ell = \mathrm{Proj}_{S^{d_\ell}}\left[w_\ell + \delta_\ell\right] \qquad \text{for } \ell = 1, \dots, L \\ & v_{L+1} = \mathrm{Proj}_{\mathcal{B}_{2,\lambda}}(w_{L+1} + \delta_{L+1}) \end{aligned}$$

where Proj_{S^d} denotes the orthogonal projection on S^d, i.e., a block coordinate normalization.

The stochastic version, starting from a given $w_{1:L+1}^{(0)}$, consists at iteration t of sampling a

function f_i and performing the step

$$w_{1:L+1}^{(t+1)} = \mathrm{ret}(-\gamma_t \nabla f_i(w_{1:L+1}^{(t)}); w_{1:L+1}^{(t)}). \tag{SGO}$$

Recall that, when solving a smooth optimization problem with manifold constraints, the relevant first-order quantity to prove the convergence to a stationarity point is the projection of the gradient on the tangent space at the current point (Boumal et al., 2016). Formally, for $w \in S$, denote by $\mathcal{T}_{S,w}$ the tangent space at w on S. Note that, for $S^{d_\ell} = \prod_{j=1}^{f_\ell} \mathbb{S}^{s_\ell}$, the tangent space is the product of the tangent spaces and on a sphere $\mathbb{S}$, $\mathcal{T}_{\mathbb{S},x} = x + V(x)^\perp$ where $V(x) = \{\alpha x; \alpha \in \mathbb{R}\}$ is the line generated by x. The convergence to a stationary point involves

$$\begin{aligned} & \operatorname{grad} f(w_{1:L+1}) = (g_1; \ldots; g_{L+1}) \\ \text{where} \qquad & g_\ell = \mathrm{Proj}_{\mathcal{T}_{S^{d_\ell}, w_\ell}}(\nabla_{w_\ell} f(w_{1:L+1})) \qquad \text{for } \ell \in \{1, \ldots, L\} \\ & g_{L+1} = \mathrm{Proj}_{\mathcal{T}_{\mathcal{B}_{2,\lambda}, w_{L+1}}}(\nabla_{w_{L+1}} f(w_{1:L+1})) \end{aligned} \tag{A.13}$$

where $\mathcal{T}_{\mathcal{B}_{2,\lambda}, w_{L+1}} = \mathbb{R}^{d_{L+1}}$ if $w_{L+1} \in \overset{\circ}{\mathcal{B}}_{2,\lambda}$ or $\mathcal{T}_{\mathcal{B}_{2,\lambda}, w_{L+1}} = \mathcal{T}_{\mathbb{S}_{2,\lambda}^{d_{L+1}}, w_{L+1}}$ if $w_{L+1} \in \mathcal{B}_{2,\lambda} \setminus \overset{\circ}{\mathcal{B}}_{2,\lambda} = \mathbb{S}_{2,\lambda}^{d_{L+1}}$.

The ingredients to prove convergence to a stationary point are then similar to the unconstrained case:

(i) Upper quadratic bounds for the steps, i.e., there exists M such that for all iterates $w_{1:L+1}^{(t)} \in \mathcal{C}$,

$$f(\mathrm{ret}(\delta_{1:L+1}; w_{1:L+1}^{(t)})) \leq f(w_{1:L+1}^{(t)}) + \operatorname{grad} f(w_{1:L+1}^{(t)})^\top \delta_{1:L+1} + \frac{M}{2} \|\delta_{1:L+1}\|_2^2,$$

where $\delta_{1:L+1} \in \mathcal{T}_{\mathcal{C}, w_{1:L+1}^{(t)}}$, the tangent space of $\mathcal{C}$ at $w_{1:L+1}^{(t)}$.

(ii) Bounded gradients, i.e., there exists B such that for all iterates $w_{1:L+1}^{(t)} \in \mathcal{C}$,

$$\|\nabla \operatorname{grad} f_i(w_{1:L+1}^{(t)})\| \leq B.$$

The convergence result is then as follows.

Theorem 56 (Hosseini and Sra, 2020, Theorem 5)**.** *Assume conditions (i) and (ii) hold. Then, for $\gamma_t = c/\sqrt{T}$ with $c > 0$ and T the maximal number of iterations, the iterates of* (SGO) *satisfy*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[\| \operatorname{grad} f(w_{1:L+1}^{(t)})\|_2^2] \leq \frac{1}{\sqrt{T}}\left(\frac{f(w_{1:L+1}^{(1)}) - f^*}{c} + \frac{Mc}{2}B^2\right).$$

Local Lipschitz and smoothness conditions are sufficient to ensure conditions (i) and (ii) on compact manifolds (Boumal et al., 2016). The convergence result then follows directly.

Proposition 57. *Assume the loss in the constrained training problem* (2.4) *and the kernel defining the network* (2.3) *are continuously differentiable. A projected stochastic gradient descent with stepsize $\gamma_t = c/\sqrt{T}$ where $c > 0$ and T is the maximal number of iterations, finds an $O(1/\sqrt{T})$-stationary point.*

A.7 Ultimate Layer Reversal

In this section we provide additional details related to the Ultimate Layer Reversal method.

A.7.1 Simplified objective

We recall how the simplified objective and the original one are related in general through the following proposition.

Proposition 58. *Assume that $f(W, V)$ is twice differentiable and that for any W, the partial functions $V \mapsto f(W, V)$ are strongly convex. Then the simplified objective $\hat{f}(W) =$*

$\min_V f(W,V)$ *is differentiable and satisfies*

$$\|\nabla \hat{f}(W)\|_2 = \|\nabla f(W, V^*)\|_2,$$

where $V^* = \arg\min_V f(W,V)$.

Proof. The pairs (W, V^*) such that $V^* \in \arg\min_V f(W,V)$ are solutions of the first order optimality condition $\nabla_V f(W,V) = 0$. By the implicit function theorem, using that $\nabla_V^2 f(W,V)$ is invertible for any pair (W,V) by strong convexity of $V \to f(W,V)$, the mapping $V^*(W) = \arg\min_V f(W,V)$ is differentiable. The simplified objective then reads $\hat{f}(W) = f(W, V^*(W))$. It is differentiable as a composition of differentiable functions and satisfies $\nabla \hat{f}(W) = \nabla_W f(W, V^*(W))$. On the other hand, $\nabla f(W, V^*) = \nabla_W f(W, V^*)$ for any $V^* = \arg\min_V f(W,V)$ such that $\nabla_V f(W, V^*) = 0$ and therefore $\|\nabla \hat{f}(W)\|_2 = \|\nabla f(W, V^*)\|_2$. □

A.7.2 Square loss

We now detail the computations of the ultimate layer reversal for the case of the square loss, for which the minimization can be performed analytically. Denote by $V \in \mathbb{R}^{d \times K}, c \in \mathbb{R}^K$ the affine classifier in the ultimate layer and $W = W_{1:L}$ the inner weights with $d = d_L$ the dimension of the penultimate layer. The objective then reads

$$\min_{V,c,W} \frac{1}{n} \sum_{i=1}^{n} (y_i - F(W; x_i)^\top V - c)^2 + \lambda \|V\|_F^2,$$

which can be compactly written as

$$\min_{W,V,c} f(W,V,c) \coloneqq \left\{ \frac{1}{n} \|Y - F(W)V - \mathbf{1}_n c^\top\|_F^2 + \lambda \|V\|_F^2 \right\}, \tag{A.14}$$

where $Y \in \mathbb{R}^{n \times K}$ is the matrix of labels, $F(W) \in \mathbb{R}^{n \times d}$ is the output of the inner layers applied to the inputs $x_1, \ldots, x_n$, and $\lambda > 0$ is a regularization parameter.

The derivation of the Ultimate Layer Reversal then follows by this proposition.

Proposition 59. *The simplified objective* $\tilde{f}(W) = \min_{V,c} f(W, V, c)$ *of* f *in* (A.14) *reads*

$$\begin{aligned}\tilde{f}(W) &= \frac{1}{n}\|\Pi Y\|_F^2 - \frac{1}{n}\,\mathbf{Tr}\left(Y^\top \Pi F(W)\big(F(W)^\top \Pi F(W) + n\lambda\,\mathrm{I}_d\big)^{-1} F(W)^\top \Pi Y\right) \\ &= \frac{1}{n}\,\mathbf{Tr}\left(Y^\top \Pi\big(\mathrm{I}_n + (n\lambda)^{-1}\Pi F(W)F(W)^\top \Pi\big)^{-1}\Pi Y\right),\end{aligned}$$

where $\Pi = I - \frac{1}{n}\mathbb{1}\mathbb{1}^\top$.

Proof. For fixed W, the optimal classifier parameters $V^*(W), c^*(W)$ can be found analytically. Minimization in c amounts to centering the labels and the inputs to the ultimate layer, i.e.,

$$c^*(W) = \frac{1}{n}(Y - F(W)V^*(W))^T\mathbb{1},$$

where $\mathbb{1}$ is a vector of ones. Define the centering matrix $\Pi = \mathrm{I} - \frac{1}{n}\mathbb{1}\mathbb{1}^\top$, an orthogonal projector. Then minimization in V gives

$$V^*(W) = \big(F(W)^\top \Pi F(W) + n\lambda\,\mathrm{I}_d\big)^{-1} F(W)^\top \Pi Y$$

and the resulting objective is

$$\begin{aligned}\tilde{f}(W) &= \frac{1}{n}\|\Pi Y\|_F^2 - \frac{1}{n}\,\mathbf{Tr}\left(Y^\top \Pi F(W)\big(F(W)^\top \Pi F(W) + n\lambda\,\mathrm{I}_d\big)^{-1} F(W)^\top \Pi Y\right) \\ &= \frac{1}{n}\,\mathbf{Tr}\left(Y^\top \Pi\big(\mathrm{I}_n + (n\lambda)^{-1}\Pi F(W)F(W)^\top \Pi\big)^{-1}\Pi Y\right),\end{aligned}$$

where the second line is obtained from the Sherman–Morrison–Woodbury formula. □

A.8 Matérn kernel

In the experiments we consider using the Matérn kernel rather than the RBF kernel. The Matérn kernel, a generalization of the RBF kernel, is defined as follows:

Definition 60. *For* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ *the* Matérn kernel *with parameters* $\sigma > 0$ *and* $\nu > 0$ *is given by*

$$\kappa_{\sigma,\nu}(\mathbf{x}, \mathbf{y}) \coloneqq G_\nu \left(\frac{\sqrt{2\nu}\|x - y\|}{\sigma} \right), \quad \textit{with} \quad G_\nu(z) \coloneqq \frac{1}{2^{\nu-1}\Gamma(\nu)} z^\nu K_\nu(z) \,,$$

where K_ν *is the modified Bessel function of the second kind of order* ν, Γ *is the Gamma function, and* $\|.\|$ *is the Euclidean norm.*

For unitary vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ *the* Matérn kernel on the sphere *with parameters* $\sigma > 0$ *and* $\nu > 0$ *is given by*

$$\kappa_{\sigma,\nu}(\mathbf{x}, \mathbf{y}) = k_{\sigma,\nu}(\langle \mathbf{x}, \mathbf{y} \rangle) \coloneqq G_\nu \left(\frac{2\sqrt{\nu(1 - \langle x, y \rangle)}}{\sigma} \right). \tag{A.15}$$

The Gaussian RBF kernel can be obtained in the limiting case as $\nu \to \infty$. An integral representation of K_ν is given for $z > 0$ and $\nu \in \mathbb{R}$ by Abramowitz and Stegun (1964, Equation 9.6.24):

$$K_\nu(z) = \int_0^\infty e^{-z \cosh(t)} \cosh(\nu t) dt,$$

which readily implies $K_\nu(z) = K_{-\nu}(z)$. The limit for $z \to 0$ is given for $\nu > 0$ by Abramowitz and Stegun (1964, Equation 9.6.9):

$$K_\nu(z) \sim \frac{1}{2}\Gamma(\nu) \left(\frac{z}{2} \right)^{-\nu}, \tag{A.16}$$

which shows that we can define continuously $G_\nu(0) = 1$.

The Matérn kernel simplifies for $\nu = p + 1/2$, $p \in \mathbb{N}$ as a product of a polynomial and an exponential (Rasmussen and Williams, 2006, Chapter 4, Equation 4.16).

Lemma 61. *For* $p \in \mathbb{N}$ *and* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$*, the half-integer Matérn kernels are given by*

$$\kappa_{\sigma,p+1/2}(\mathbf{x}, \mathbf{y}) = G_{p+1/2} \left(\frac{\sqrt{2p+1}\|x - y\|}{\sigma} \right),$$

with $G_{p+1/2}(z) = \exp(-z)\frac{p!}{(2p)!}\sum_{j=0}^{p}\frac{(2p-j)!}{j!(p-j)!}(2z)^j$.

This gives for example

$$G_{3/2}(z) = (1+z)\exp(-z)$$
$$\text{and} \qquad G_{5/2}(z) = \left(1+z+\frac{z^2}{3}\right)\exp(-z)\ .$$

These are the most commonly used versions of the Matérn kernel (Rasmussen and Williams, 2006, Chapter 4).

In the general case we have the following lemma.

Lemma 62. *The spherical Matérn kernel* $k_{\sigma,\nu}$ (A.15) *is differentiable for* $\sigma > 0$ *and* $\nu > 1$. *Specifically, we have*

$$k'_{\sigma,\nu}(\langle \mathbf{x}, \mathbf{y}\rangle) = \frac{1}{\sigma'^2}k_{\sigma',\nu-1}(\langle \mathbf{x}, \mathbf{y}\rangle)\ ,$$

with $\sigma' = \sigma\sqrt{\nu-1}/\sqrt{\nu}$.

Proof. Given unitary vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, the spherical Matérn kernel reads

$$k_{\sigma,\nu}(\theta) = G_\nu\left(\frac{2\sqrt{\nu(1-\theta)}}{\sigma}\right)\ ,$$

where $\theta = \langle \mathbf{x}, \mathbf{y}\rangle \in [-1, 1]$. For $\theta < 1$, denoting $f(\theta) = \frac{2\sqrt{\nu(1-\theta)}}{\sigma}$, we get

$$k'_{\sigma,\nu}(\theta) = f'(\theta)G'_\nu(f(\theta))$$

with $f'(\theta) = -\sqrt{\nu}/(\sigma\sqrt{1-\theta})$ and

$$\begin{aligned} G'_\nu(z) &= \frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\nu z^{\nu-1}K_\nu(z) + z^\nu K'_\nu(z)\right) \\ &= \frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\nu z^{\nu-1}K_\nu(z) - z^\nu\left(K_{\nu-1}(z) + \frac{\nu}{z}K_\nu(z)\right)\right) \end{aligned}$$

$$= -\frac{z^\nu K_{\nu-1}(z)}{2^{\nu-1}\Gamma(\nu)} \tag{A.17}$$
$$= -\frac{z}{2(\nu-1)} G_{\nu-1}(z) \,,$$

where we used $K'_\nu(z) = -K_{\nu-1}(z) - \frac{\nu}{z}K_\nu(z)$ (Abramowitz and Stegun, 1964, Equation 9.6.26). Therefore

$$k'_{\sigma,\nu}(\theta) = \frac{\nu}{(\nu-1)\sigma^2} G_{\nu-1}\left(\frac{2\sqrt{\nu(1-\theta)}}{\sigma}\right).$$

The derivative for $\theta = 1$ can then be defined continuously. □

Note that for $0 < \nu < 1$ it can easily be shown from (A.17) and (A.16) that the derivative will diverge for $\langle \mathbf{x}, \mathbf{y} \rangle \to 1$.

A.9 Details on Training Methods and Additional Results

In this appendix we report additional experimental details and results. First, we compare approximations of the random features kernel (2.1) when a is the ReLU nonlinearity and the $w \sim N(0, \mathrm{I}_{25\times 25})$. We do this on 5×5 patches from standardized MNIST images when varying the dimension of the approximations. Figure A.6 compares the results from using (1) the random features $a(W, x)$ where each entry of W is drawn from $N(0, 1)$; (2) the Nyström method where the rows of W are chosen by randomly sampling from a set of 10,000 patches; and (3) the Nyström method where the rows of W are chosen as the centroids output when running spherical k-means on a set of 10,000 patches. The relative error is computed as $\|\hat{K} - K\|_F / \|K\|_F$, where K is the true Gram matrix on the 10,000 patches and $\hat{K}$ is the approximated Gram matrix computed using one of the above approximation methods. From the results we can see that using the Nyström method with spherical k-means to select W performs the best. The random features method performs 6.8 to 650 times worse for a given number of features, while the Nyström method with random sampling to select W performs 1.3 to 4.0 times worse.

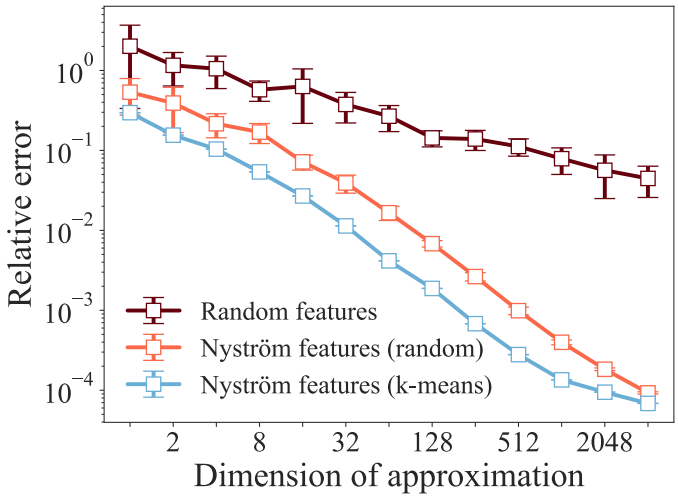


Figure A.6: Average performance of three kernel approximation methods across 10 trials when varying the dimension of the approximation. The error bars show one standard deviation from the mean.

Next, Table A.2 provides the batch size we use for each ConvNet and CKN experiment. The batch size is the largest size that fits on the GPU. This batch size declines as the size of the architecture increases since larger architectures consume more memory.

We report in Figure A.7 the results of using the proposed Ultimate Layer Reversal method in terms of loss vs. time. We do so for the LeNet-5 CKN with 8 and 128 filters/layer and for the All-CNN-C CKN on CIFAR-10 with 8 and 128 filters/layer. We can see from the plots that the Ultimate Layer Reversal method, ULR-SGO, outperforms stochastic gradient optimization, SGO, on the more difficult tasks.

Next, Figure A.8 shows the performance of the unsupervised initialization of the CKNs relative to the trained CKNs and ConvNets. We can see that the unsupervised performance improves as the number of filters per layer increases, as we would expect. For the LeNets the performance varies between 61% and 99% of the performance of the supervised CKN. For All-CNN-C and AlexNet the results are more modest: on All-CNN-C the unsupervised performance is 41-55% of that of the supervised CKN and on AlexNet the unsupervised

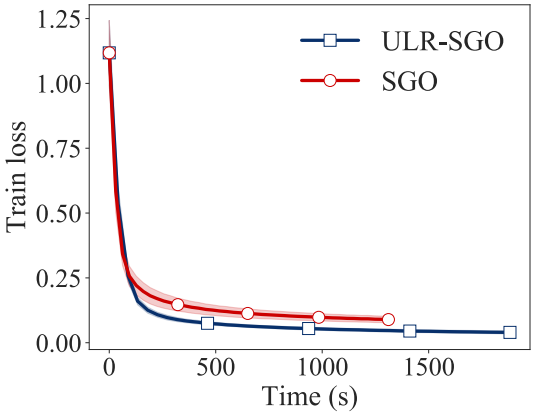
	Number of filters per layer				
Architecture	8	16	32	64	128
LeNet-1	16384	8192	4096	2048	1024
LeNet-5	16384	8192	4096	2048	1024
All-CNN-C	4096	2048	1024	512	256
AlexNet	512	512	512	256	128

Table A.2: Batch sizes used for the LeNet-1, LeNet-5, All-CNN-C, and AlexNet CKNs and ConvNets.

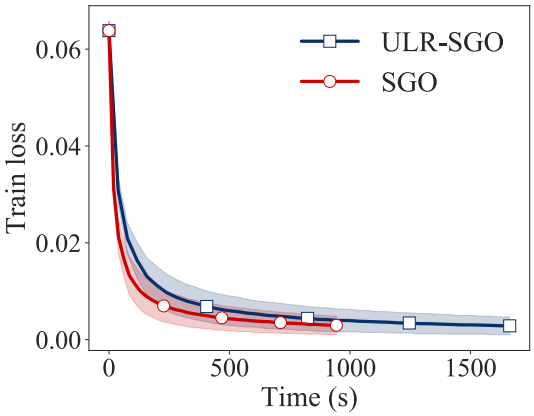
performance is 34-41% of that of the supervised CKN. This suggests that for more complex tasks it is more difficult to obtain a good unsupervised performance.

Figure A.9 explores the effect of using the Matérn kernel rather than the Gaussian RBF kernel when training the AlexNet CKN on ImageNet. The hold-out validation in this case is performed for the first 2000 iterations over the order of the kernel, the kernel bandwidth, the regularization parameter τ of the Hessian, and the step size. The parameter values considered are 1.5, 2.5, and 3.5 for the order of the kernel, $0.4, 0.5, \ldots, 0.9$. for the bandwidth, 2^i for $i = -6, -5, -4$ for τ, and 2^i for $i = -8, -7, \ldots, -4$ for the step size. After the first 2000 iterations the hold-out validation proceeds as described in Section 2.5.1. From the plots we can see that the CKN with the Matérn kernel performs similarly to the CKN with the Gaussian RBF kernel. In particular, the CKN with the Matérn kernel does between 6% worse (in the case of 64 filters/layer) and 9% better (in the case of 8 filters/layer).

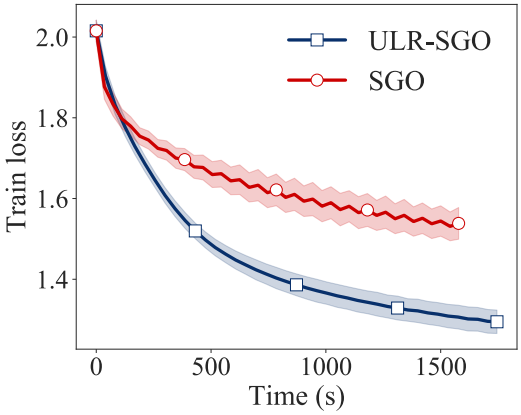
Finally, Tables A.3 and A.4 provide the numerical values of the means and standard deviations reported in Figures 2.4, A.8, and A.9.



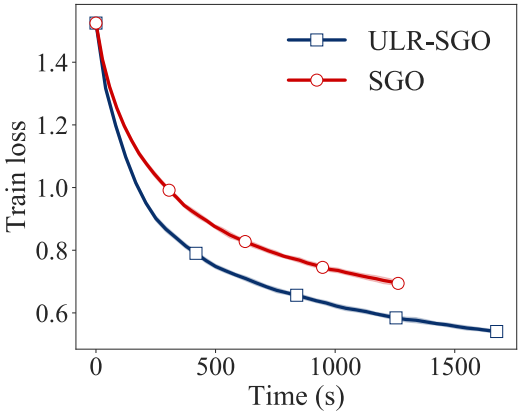
(a) LeNet-5 CKN on MNIST with 8 filters/layer



(b) LeNet-5 CKN on MNIST with 128 filters/layer



(c) All-CNN-C CKN on CIFAR-10 with 8 filters/layer



(d) All-CNN-C CKN on CIFAR-10 with 128 filters/layer

Figure A.7: Performance of CKNs when using plain stochastic gradient optimization (SGO) vs. the Ultimate Layer Reversal method (ULR-SGO) in terms of the training loss vs. time. The error bands represent one standard deviation across 10 trials with different random seeds.

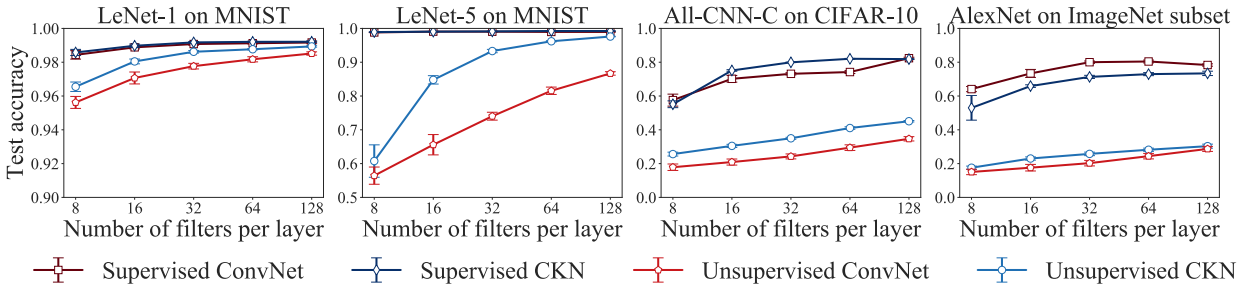


Figure A.8: Average performance of ConvNets and their CKN counterparts across 10 trials when varying the number of filters per layer. Note that the y-axis scales differ across the plots. The error bars show one standard deviation from the mean.

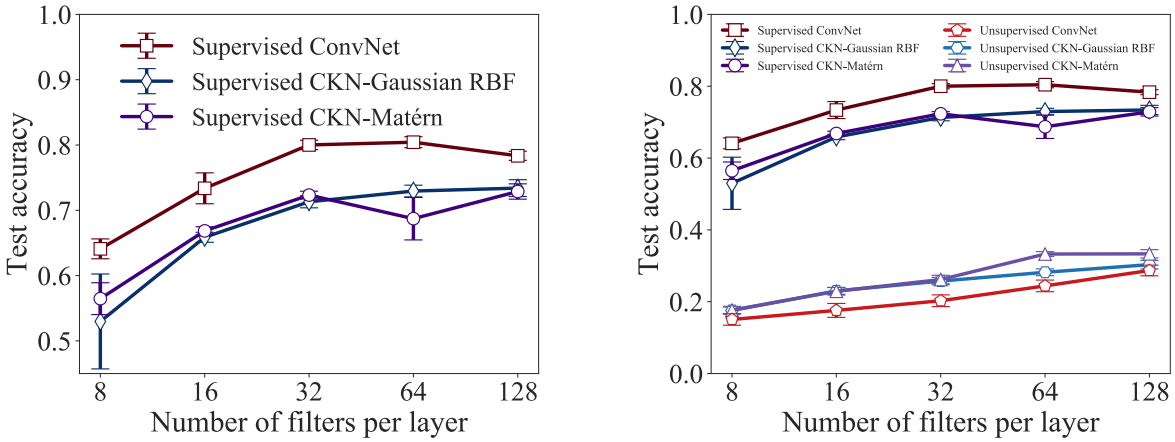


Figure A.9: Average performance of ConvNets and their CKN counterparts across 10 trials when training versions of AlexNet on a subset of ImageNet and varying the number of filters per layer. The error bars show one standard deviation from the mean.

Architecture	Number of filters per layer				
	8	16	32	64	128
LeNet-1 CKN	0.986 (0.002)	0.990 (0.001)	0.992 (0.001)	0.992 (0.000)	0.992 (0.001)
LeNet-1 ConvNet	0.984 (0.003)	0.989 (0.001)	0.991 (0.001)	0.991 (0.000)	0.992 (0.001)
LeNet-5 CKN	0.989 (0.001)	0.991 (0.001)	0.991 (0.001)	0.992 (0.001)	0.992 (0.001)
LeNet-5 ConvNet	0.988 (0.001)	0.991 (0.001)	0.990 (0.001)	0.990 (0.001)	0.990 (0.001)
All-CNN-C CKN	0.551 (0.020)	0.751 (0.005)	0.799 (0.001)	0.820 (0.002)	0.819 (0.003)
All-CNN-C ConvNet	0.576 (0.035)	0.702 (0.020)	0.731 (0.007)	0.742 (0.009)	0.824 (0.006)
AlexNet CKN	0.530 (0.073)	0.659 (0.008)	0.713 (0.009)	0.730 (0.009)	0.734 (0.013)
AlexNet CKN-Matérn	0.565 (0.024)	0.669 (0.006)	0.723 (0.006)	0.687 (0.033)	0.729 (0.012)
AlexNet ConvNet	0.641 (0.015)	0.734 (0.024)	0.800 (0.007)	0.804 (0.008)	0.783 (0.007)

Table A.3: Average supervised accuracies from Figures 2.4 and A.9 with the corresponding standard deviations in parentheses. The means and standard devations are computed across 10 trials with different random seeds.

Architecture	Number of filters per layer				
	8	16	32	64	128
LeNet-1 CKN	0.965 (0.003)	0.980 (0.001)	0.986 (0.001)	0.988 (0.000)	0.989 (0.000)
LeNet-1 ConvNet	0.956 (0.004)	0.971 (0.004)	0.978 (0.002)	0.982 (0.002)	0.985 (0.001)
LeNet-5 CKN	0.608 (0.048)	0.848 (0.012)	0.933 (0.005)	0.962 (0.002)	0.976 (0.001)
LeNet-5 ConvNet	0.564 (0.025)	0.656 (0.030)	0.740 (0.012)	0.816 (0.011)	0.867 (0.005)
All-CNN-C CKN	0.256 (0.011)	0.305 (0.009)	0.350 (0.006)	0.411 (0.007)	0.451 (0.004)
All-CNN-C ConvNet	0.179 (0.019)	0.209 (0.018)	0.243 (0.015)	0.295 (0.017)	0.347 (0.013)
AlexNet CKN	0.176 (0.010)	0.230 (0.010)	0.258 (0.010)	0.282 (0.009)	0.303 (0.012)
AlexNet CKN-Matérn	0.177 (0.011)	0.229 (0.011)	0.262 (0.012)	0.333 (0.007)	0.333 (0.012)
AlexNet ConvNet	0.151 (0.016)	0.176 (0.019)	0.203 (0.016)	0.244 (0.016)	0.287 (0.015)

Table A.4: Average unsupervised accuracies from Figures A.8 and A.9 with the corresponding standard deviations in parentheses. The means and standard devations are computed across 10 trials with different random seeds.

Appendix B

APPENDIX FOR CHAPTER 3

B.1 Smoothness of the Objective Function

In this appendix we estimate the smoothness constants for "forward prediction" regularized least squares and "backward prediction" least squares. Regularized forward prediction least squares learns to predict the label matrix Y from the features Φ:

$$\min_W \frac{1}{n}\|Y - \Phi W - \mathbb{1}_n b^T\|_F^2 + \lambda\|W\|_F^2$$

In contrast, reverse prediction least squares learns to predict the features Φ from the labels Y:

$$\min_W \frac{1}{n}\|\Phi - YW\|_F^2 \,.$$

As noted by Xu et al. (2009), the solution of the forward prediction problem can be recovered from the solution of the reverse prediction problem as long as Φ is full rank.

Now we return to Proposition 7, which compared the Lipschitz constants of the two objectives, and provide its proof.

Proposition 7. *Let $\mathcal{Z}$ be the set of all possible feature matrices $\Phi \in \mathbb{R}^{n\times D}$. Assume there exists $B \in \mathbb{R}$ such that for all $\Phi \in \mathcal{Z}$, $\|\Phi\|_2 \le B$. Let $n_{\max}$ be a bound on the maximum number of points in a cluster. Then the Lipschitz constants of F_f and F_r with respect to the spectral norm can be estimated by*

$$L_f \coloneqq 2B\left(\frac{n_{\max}}{n}\right)\left(\frac{1}{n\lambda}\right) \quad \text{and} \quad L_r \coloneqq \frac{2}{n}B \,,$$

respectively. We therefore have $L_f \leq L_r$ *for* $\lambda \geq n_{\max}/n$.

Proof. After minimizing in the classifier variable W, the forward prediction objective reads

$$F_f(\Phi) = \lambda \operatorname{trace}[YY^T\Pi_n(\Pi_n\Phi\Phi^T\Pi_n + n\lambda\,\mathrm{I})^{-1}\Pi_n] \,.$$

Define $G(\Phi) = \left(\Pi_n\Phi\Phi^T\Pi_n + n\lambda\,\mathrm{I}_n\right)^{-1}$. The gradient of F_f is then

$$\nabla F_f(\Phi) = -2\lambda\Pi_n G(\Phi)\Pi_n YY^T\Pi_n G(\Phi)\Pi_n\Phi.$$

Since $\|G(\Phi)\|_2 \leq 1/(n\lambda)$, $\|YY^T\|_2 \leq n_{\max}$, $\|\Pi_n\|_2 \leq 1$ and $\|G(\Phi)\Pi_n\Phi\|_2 \leq \|\Pi_n\Phi\|_2/(n\lambda)$, we obtain

$$\|\nabla F_f(\Phi)\|_2 \leq 2\frac{n_{\max}}{\lambda n^2}B =: L_f.$$

Next, recall that the reverse prediction objective for fixed cluster assignments Y may be written as $F_r(\Phi) = \frac{1}{n}\operatorname{trace}[(\mathrm{I} - P_Y)\Phi\Phi^T]$ where $P_Y = Y(Y^TY)^{-1}Y^T$ is an orthonormal projector. Its gradient, $\nabla F_r(\Phi) = \frac{2}{n}(\mathrm{I} - P_Y)\Phi$, can therefore be bounded as

$$\|\nabla F_r(\Phi)\|_2 \leq \frac{2}{n}B =: L_r.$$

Hence, taking $\lambda \geq n_{\max}/n$, we have $L_f \leq L_r$. □

Before moving on to the smoothness of the gradient we prove a lemma. The lemma estimates the Lipschitz constant of the gradient of the "forward prediction" objective function $F_f(\Phi)$ from Section 3.4.1.

Lemma 63. *Consider a feature matrix* $\Phi \in \mathcal{Z}$ *with* $\Phi \in \mathbb{R}^{n\times D}$ *and define the function* $F_f : \mathbb{R}^{n\times D} \to \mathbb{R}$ *by* $F_f(\Phi) = \lambda \operatorname{trace}[YY^T\Pi_n(\Pi_n\Phi\Phi^T\Pi_n + n\lambda\,\mathrm{I})^{-1}\Pi_n]$. *Assume there exists* B

such that for all $\Phi \in \mathcal{Z}$, $\|\Phi\|_2 \leq B$. *Then for all* $\Phi_1, \Phi_2 \in \mathcal{Z}$,

$$\|\nabla F_f(\Phi_1) - \nabla F_f(\Phi_2)\|_2 \leq \left(\frac{8B^2 n_{\max}}{n^3\lambda^2} + \frac{2n_{\max}}{n^2\lambda}\right) \|\Phi_1 - \Phi_2\|_2 \, .$$

Hence, an upper bound on the Lipschitz constant of the gradient of $F_f(\Phi)$ *is given by*

$$\ell_f \coloneqq \frac{2n_{\max}}{n^2\lambda} + \frac{8B^2 n_{\max}}{n^3\lambda^2} \, .$$

Proof. Note that the gradient of F_f is given by $\nabla F_f(\Phi) = -2\lambda \Pi_n G(\Phi)\Pi_n Y Y^T \Pi_n G(\Phi)\Pi_n \Phi$, where $G(\Phi) = (\Pi_n \Phi\Phi^T \Pi_n + n\lambda I)^{-1}$. Now let $\|\cdot\| = \|\cdot\|_2$ denote the spectral norm and observe that, using that $\|\Pi_n\| \leq 1$ since Π_n is a projection matrix,

$$\begin{aligned}
&\frac{1}{2\lambda}\|\nabla F_f(\Phi_1) - \nabla F_f(\Phi_2)\| \\
&= \left\|\Pi_n G(\Phi_1)\Pi_n Y Y^T \Pi_n G(\Phi_1)\Pi_n \Phi_1 - \Pi_n G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\Pi_n \Phi_2\right\| \\
&\leq \underbrace{\left\|G(\Phi_1)\Pi_n Y Y^T \Pi_n G(\Phi_1)\Pi_n \Phi_1 - G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\Pi_n \Phi_1\right\|}_{(a)} \\
&\quad + \underbrace{\left\|G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\Pi_n \Phi_1 - G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\Pi_n \Phi_2\right\|}_{(b)} \, .
\end{aligned}$$

First consider term (a). We have that

$$\begin{aligned}
&\left\|G(\Phi_1)\Pi_n Y Y^T \Pi_n G(\Phi_1)\Pi_n \Phi_1 - G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\Pi_n \Phi_1\right\| \\
&= \left\|[G(\Phi_1)\Pi_n Y Y^T \Pi_n G(\Phi_1) - G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)]\Pi_n \Phi_1\right\| \\
&\leq \underbrace{\left\|G(\Phi_1)\Pi_n Y Y^T \Pi_n G(\Phi_1) - G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\right\|}_{(c)} \|\Phi_1\| \, .
\end{aligned}$$

We may bound term (c) by

$$\left\|G(\Phi_1)\Pi_n Y Y^T \Pi_n G(\Phi_1) - G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\right\|$$

$$
\begin{aligned}
&\leq \left\|G(\Phi_1)\Pi_n YY^T\Pi_n G(\Phi_1) - G(\Phi_1)\Pi_n YY^T\Pi_n G(\Phi_2)\right\| \\
&\quad + \left\|G(\Phi_1)\Pi_n YY^T\Pi_n G(\Phi_2) - G(\Phi_2)\Pi_n YY^T\Pi_n G(\Phi_2)\right\| \\
&\leq \left\|G(\Phi_1)\Pi_n YY^T\Pi_n\right\| \underbrace{\|G(\Phi_1) - G(\Phi_2)\|}_{(d)} + \left\|G(\Phi_2)\Pi_n YY^T\Pi_n\right\| \|G(\Phi_1) - G(\Phi_2)\| \ .
\end{aligned}
$$

Furthermore, we can bound term (d) via

$$
\begin{aligned}
\|G(\Phi_1) - G(\Phi_2)\| &= \left\|G(\Phi_1)\left[G(\Phi_1)^{-1} - G(\Phi_2)^{-1}\right]G(\Phi_2)\right\| \\
&\leq \|G(\Phi_1)\|\,\|G(\Phi_2)\| \underbrace{\left\|G(\Phi_1)^{-1} - G(\Phi_2)^{-1}\right\|}_{(e)}
\end{aligned}
$$

Finally, we can bound term (e) by

$$
\begin{aligned}
\left\|G(\Phi_1)^{-1} - G(\Phi_2)^{-1}\right\| &= \left\|\Pi_n\Phi_1\Phi_1^T\Pi_n - \Pi_n\Phi_2\Phi_2^T\Pi_n\right\| \\
&\leq \left\|\Phi_1\Phi_1^T - \Phi_1\Phi_2^T\right\| + \left\|\Phi_1\Phi_2^T - \Phi_2\Phi_2^T\right\| \\
&\leq \|\Phi_1\|\|\Phi_1 - \Phi_2\| + \|\Phi_2\|\|\Phi_1 - \Phi_2\| \ .
\end{aligned}
$$

Using this above, a bound on term (a) is thus

$$
\begin{aligned}
&\left\|G(\Phi_1)\Pi_n YY^T\Pi_n G(\Phi_1)\Pi_n\Phi_1 - G(\Phi_2)\Pi_n YY^T\Pi_n G(\Phi_2)\Pi_n\Phi_1\right\| \\
\leq\ &\left(\left\|G(\Phi_1)\Pi_n YY^T\Pi_n\right\| + \left\|G(\Phi_2)\Pi_n YY^T\Pi_n\right\|\right)\|G(\Phi_1)\|\,\|G(\Phi_2)\|\,(\|\Phi_1\| + \|\Phi_2\|) \\
&\times \|\Phi_1\|\|\Phi_1 - \Phi_2\| \ .
\end{aligned}
$$

Next, consider term (b). We have that

$$
\begin{aligned}
&\left\|G(\Phi_2)\Pi_n YY^T\Pi_n G(\Phi_2)\Pi_n\Phi_1 - G(\Phi_2)\Pi_n YY^T\Pi_n G(\Phi_2)\Pi_n\Phi_2\right\| \\
\leq &\left\|G(\Phi_2)\Pi_n YY^T\Pi_n G(\Phi_2)\right\|\|\Phi_1 - \Phi_2\| \ .
\end{aligned}
$$

Therefore, returning to the original quantity of interest, we have

$$\begin{aligned}
&\frac{1}{2\lambda}\left\|\nabla F_f(\Phi_1) - \nabla F_f(\Phi_2)\right\| \\
&\leq \left\{\left(\left\|G(\Phi_1)\Pi_n Y Y^T \Pi_n\right\| + \left\|G(\Phi_2)\Pi_n Y Y^T \Pi_n\right\|\right)\|G(\Phi_1)\|\,\|G(\Phi_2)\|\,(\|\Phi_1\| + \|\Phi_2\|)\,\|\Phi_1\| \right. \\
&\quad \left. + \left\|G(\Phi_2)\Pi_n Y Y^T \Pi_n G(\Phi_2)\right\|\right\} \times \|\Phi_1 - \Phi_2\| \,.
\end{aligned}$$

We have $\|YY^T\|_2 \leq n_{\max}$, where $n_{\max}$ is a bound on the maximum size of the clusters. Lastly, $\|G(\Phi_1)\|_2 \leq 1/(n\lambda)$ and $\|G(\Phi_2)\|_2 \leq 1/(n\lambda)$. Therefore, we have

$$\begin{aligned}
\frac{1}{2\lambda}\left\|\nabla F_f(\Phi_1) - \nabla F_f(\Phi_2)\right\|_2 &\leq \left\{\frac{2n_{\max}}{n\lambda}\left(\frac{1}{n\lambda}\right)^2 (2B)B + \frac{n_{\max}}{n^2\lambda^2}\right\}\|\Phi_1 - \Phi_2\|_2 \\
&= \left(\frac{4B^2 n_{\max}}{n^3\lambda^3} + \frac{n_{\max}}{n^2\lambda^2}\right)\|\Phi_1 - \Phi_2\|_2 \,,
\end{aligned}$$

and so an upper bound on the Lipschitz constant is given by

$$\ell_f \coloneqq \frac{2n_{\max}}{n^2\lambda} + \frac{8B^2 n_{\max}}{n^3\lambda^2} \,.$$

□

Now we recall Proposition 8 from Section 3.4.1, which compared the Lipschitz constants of the gradients of the forward and reverse prediction objectives, and provide its proof.

Proposition 8. *Under the same assumption as Proposition 7, the Lipschitz constants of ∇F_f and ∇F_r with respect to the spectral norm can be estimated by*

$$\ell_f \coloneqq 2\left(\frac{n_{\max}}{n}\right)\left(\frac{1}{n\lambda}\right) + 8B^2\left(\frac{n_{\max}}{n}\right)\left(\frac{1}{n^2\lambda^2}\right) \quad \textit{and} \quad \ell_r \coloneqq \frac{2}{n} \,,$$

respectively. We therefore have $\ell_f \leq \ell_r$ for $\lambda \geq n_{\max}/(2n) + \sqrt{n_{\max}^2 + 16B^2 n_{\max}}/(2n)$.

Proof. The gradient of F_f is given by $\nabla F_f(\Phi) = -2\lambda\Pi_n G(\Phi)\Pi_n Y\tilde{Y}^T\Pi_n G(\Phi)\Pi_n\Phi$. By

Lemma 63 we have that

$$\|\nabla F_f(\Phi_1) - \nabla F_f(\Phi_2)\|_2 \leq \left(\frac{2n_{\max}}{n^2\lambda} + \frac{8B^2 n_{\max}}{n^3\lambda^2}\right)\|\Phi_1 - \Phi_2\|_2 \ .$$

Next, observe that the gradient of F_r is $\nabla F_r(\Phi) = \frac{2}{n}(\mathrm{I} - P_Y)\Phi$. Hence, we have

$$\|\nabla F_r(\Phi_1) - F_r(\Phi_2)\|_2 \leq \frac{2}{n}\|\Phi_1 - \Phi_2\|_2 \ .$$

For $\lambda \geq n_{\max}/(2n) + \sqrt{n_{\max}^2 + 16B^2 n_{\max}}/(2n)$, we therefore have $\ell_f \leq \ell_r$. □

B.2 Solving the Label Assignment Problem

Now we address the problem of optimizing the labels for the unlabeled data. The following proposition shows that this discrete problem is in general NP-complete for $k > 2$.

Proposition 64. *Let $A \in \mathbb{R}^{n\times n}$. The label assignment problem*

$$\begin{aligned} &\min_Y \ \mathrm{trace}(YY^T A) \\ &s.t. \ \sum_{j=1}^{k} Y_{ij} = 1, \quad i = 1, \ldots, n \\ &\quad Y_{ij} \in \{0,1\} \quad \forall\, i = 1, \ldots, n,\ j = 1, \ldots, k \end{aligned}$$

is NP-complete for $k > 2$.

Proof. The proof will follow by showing that the k-coloring problem is a special case of the matrix balancing problem. Let G be an undirected, unweighted graph with no self-loops. Define $A \in \{0,1\}^{n\times n}$ to be the adjacency matrix of G. Then G is k-colorable if and only if the following problem has minimum value zero:

$$\min_Y \sum_{j=1}^{k} \sum_{i,i' \in A} Y_{i,j} Y_{i',j}$$

$$s.t. \sum_{j=1}^{k} Y_{ij} = 1, \quad i = 1, \ldots, n$$
$$Y_{ij} \in \{0, 1\} \quad \forall\, i = 1, \ldots, n,\ j = 1, \ldots, k\ .$$

Noting that

$$\sum_{j=1}^{k} \sum_{i,i' \in A} Y_{i,j} Y_{i',j} = \text{trace}(YY^T A)\ ,$$

we may rewrite the above problem as

$$\min_{Y}\ \text{trace}(YY^T A)$$
$$s.t. \sum_{j=1}^{k} Y_{ij} = 1, \quad \forall\, i = 1, \ldots, n$$
$$Y_{ij} \in \{0, 1\} \quad \forall\, i = 1, \ldots, n,\ j = 1, \ldots, k\ .$$

This is a special case of the matrix balancing problem, in which A is the adjacency matrix of a graph. Therefore, as the k-coloring problem is NP-complete for $k > 2$ (Karp, 1975), the label assignment problem with discrete assignments is also NP-complete for $k > 2$. □

B.2.1 Matrix balancing algorithm

Due to the previous result we optimize a convex relaxation of the label assignment problem. Consider a similarity matrix $A \in \mathbb{R}^{n \times n}$ and a matrix $M \in \mathbb{R}^{n \times n}$ with some known entries m_{ij} whose indices lie in the set $\mathcal{K} \subset \{1, \ldots, n\}^2$. Then, given minimum and maximum cluster sizes $n_{\min}$ and $n_{\max}$, the problem we solve is

$$\begin{aligned} \min_{M \in \mathbb{R}^{n \times n}} \quad & \mathbf{Tr}(M^T A) \\ \text{subject to} \quad & n_{\min} \mathbb{1} \leq M \mathbb{1} \leq n_{\max} \mathbb{1}, \quad n_{\min} \mathbb{1} \leq M^T \mathbb{1} \leq n_{\max} \mathbb{1}, \quad M \geq 0 \end{aligned}$$

$$M_{ij} = m_{ij}, \quad (i,j) \in \mathcal{K} \ ,$$

where $\mathbb{1} = \mathbb{1}_n$ is the vector of ones in $\mathbb{R}^n$. The constants $n_{\max}$ and $n_{\min}$ are upper and lower bounds on the sizes of the clusters, respectively.

We consider an entropic regularization of the problem. Specifically, we use the entropic regularizer

$$h(M) = \sum_{ij} M_{ij} \log(M_{ij}) \quad \text{with} \quad \nabla h(M) = \mathbb{1}\mathbb{1}^T + \log(M) \ ,$$

where $\log(M)$ is understood to be applied element-wise. We make a proximal step from an initial guess M_0 by considering the Bregman divergence $D_h(M; M_0) = h(M) - h(M_0) - \langle \nabla h(M_0), M - M_0 \rangle = h(M) - \langle \mathbb{1}\mathbb{1}^T + \log(M_0), M \rangle + \mathrm{C}$, where C is a constant. If $\mathcal{K} = \emptyset$ a feasible M_0 is given by $(M_0)_{ij} = 1/k$, where k is the number of clusters. Consider then the problem

$$\begin{aligned} \min_{M \in \mathbb{R}^{n \times n}} \quad & \mathbf{Tr}(M^T A) + \mu D_h(M; M_0) \\ \text{subject to} \quad & n_{\min}\mathbb{1} \leq M\mathbb{1} \leq n_{\max}\mathbb{1}, \quad n_{\min}\mathbb{1} \leq M^T\mathbb{1} \leq n_{\max}\mathbb{1}, \quad M \geq 0 \\ & M_{ij} = m_{ij}, \quad (i,j) \in \mathcal{K} \ , \end{aligned}$$

which is equivalent to

$$\begin{aligned} \min_{M \in \mathbb{R}^{n \times n}} \quad & \mathbf{Tr}(M^T Q) + \mu h(M) \\ \text{subject to} \quad & n_{\min}\mathbb{1} \leq M\mathbb{1} \leq n_{\max}\mathbb{1}, \quad n_{\min}\mathbb{1} \leq M^T\mathbb{1} \leq n_{\max}\mathbb{1}, \quad M \geq 0 \\ & M_{ij} = m_{ij}, \quad (i,j) \in \mathcal{K} \ . \end{aligned} \tag{B.1}$$

where $Q = A - \mu\mathbb{1}\mathbb{1}^T - \mu \log(M_0)$.

Dual problem. Considering the problem scaled by μ^{-1} and introducing Lagrange multipliers, we obtain

$$\begin{aligned}\max_{\alpha\in\mathbb{R}^n,\beta\in\mathbb{R}^n,\gamma\in\mathbb{R}^n,\delta\in\mathbb{R}^n,\lambda\in\mathbb{R}^{|\mathcal{K}|}}\ \min_{M\in\mathbb{R}^{n\times n}}\quad & \mu^{-1}\,\mathbf{Tr}(M^TQ)+h(M)+\alpha^T(n_{\min}\mathbb{1}-M\mathbb{1})\\ & -\beta^T(n_{\max}\mathbb{1}-M\mathbb{1})+\gamma^T(n_{\min}\mathbb{1}-M^T\mathbb{1})\\ & -\delta^T(n_{\max}\mathbb{1}-M^T\mathbb{1})+\sum_{(i,j)\in\mathcal{K}}\lambda_{ij}(M_{ij}-m_{ij})\\ \text{subject to}\quad & M\geq 0,\quad \alpha\geq 0,\quad \beta\geq 0,\quad \gamma\geq 0,\quad \delta\geq 0\,.\end{aligned}$$

The minimum in M for $\alpha,\beta,\gamma,\delta$, and λ fixed can be computed analytically. It reads

$$M^\star=\exp(-(\tilde{Q}+(\beta-\alpha)\mathbb{1}^T+\mathbb{1}(\delta-\gamma)^T+\Lambda))\,,$$

where $\tilde{Q}=\mu^{-1}Q+\mathbb{1}\mathbb{1}^T=\mu^{-1}A-\log(M_0)$ and $\Lambda=[\Lambda_{ij}]_{i,j=1}^n$ with $\Lambda_{ij}=\lambda_{ij}$ if $(i,j)\in\mathcal{K}$ and $\Lambda_{ij}=0$ otherwise. The dual problem then reads

$$\begin{aligned}\min_{\alpha\in\mathbb{R}^n,\beta\in\mathbb{R}^n,\gamma\in\mathbb{R}^n,\delta\in\mathbb{R}^n,\lambda\in\mathbb{R}^{|\mathcal{K}|}}\quad & \mathbb{1}^T\exp(-(\tilde{Q}+(\beta-\alpha)\mathbb{1}^T+\mathbb{1}(\delta-\gamma)^T+\Lambda))\mathbb{1}\\ & -n_{\min}(\alpha+\gamma)^T\mathbb{1}+n_{\max}(\beta+\delta)^T\mathbb{1}+\sum_{(i,j)\in\mathcal{K}}\lambda_{ij}m_{ij}\\ \text{subject to}\quad & \alpha\geq 0,\quad \beta\geq 0,\quad \gamma\geq 0,\quad \delta\geq 0\,.\end{aligned}$$

Using the change of variables $a=\beta-\alpha$, $b=\beta+\alpha$, $c=\delta-\gamma$, $d=\delta+\gamma$, the dual problem is then

$$\begin{aligned}\min_{a\in\mathbb{R}^n,b\in\mathbb{R}^n,c\in\mathbb{R}^n,d\in\mathbb{R}^n,\lambda\in\mathbb{R}^{|\mathcal{K}|}}\quad & \exp(-a)^T\exp(-(\tilde{Q}+\Lambda))\exp(-c)\\ & +\frac{n_{\max}-n_{\min}}{2}(b+d)^T\mathbb{1}+\frac{n_{\max}+n_{\min}}{2}(a+c)^T\mathbb{1}+\sum_{(i,j)\in\mathcal{K}}\lambda_{ij}m_{ij}\\ \text{subject to}\quad & b\geq|a|,\quad d\geq|c|\,.\end{aligned}$$

Minimization in b and d can be performed analytically (using $n_{\max} \geq n_{\min}$) such that the problem reads

$$\min_{a\in\mathbb{R}^n, c\in\mathbb{R}^n, \lambda\in\mathbb{R}^{|\mathcal{K}|}} \quad \exp(-a)^T \exp(-(\tilde{Q}+\Lambda)) \exp(-c) + n_\Delta(\|a\|_1 + \|c\|_1) + n_\Sigma (a+c)^T \mathbb{1} \\ + \sum_{(i,j)\in\mathcal{K}} \lambda_{ij} m_{ij} \ , \tag{B.2}$$

where $n_\Delta = (n_{\max} - n_{\min})/2$ and $n_\Sigma = (n_{\max} + n_{\min})/2$.

Alternating minimization. We consider an alternating minimization scheme to solve (B.2).

Minimization in λ. Differentiating with respect to λ leads to

$$\exp(-\lambda^\star_{ij}) = \exp(\tilde{Q}_{ij}) \exp(a_i^\star) \exp(c_j^\star) m_{ij} \ , \quad (i,j) \in \mathcal{K} \ .$$

Minimization in a. Minimization in a for c and λ fixed reads, denoting $x = \exp(-(\tilde{Q} + \Lambda^\star)) \exp(-c^\star)$,

$$\min_a \max_{\|\eta\|_\infty \leq 1} \exp(-a)^T x + a^T (n_\Sigma \mathbb{1} + n_\Delta \eta) \ .$$

Switching the max and min, the minimum in a for η fixed (which is always defined since $n_\Sigma \mathbb{1} + n_\Delta \eta \geq 0$ for $\|\eta\|_\infty \leq 1$) is given by

$$\exp(-a^\star(\eta)_i) = (n_\Sigma + n_\Delta \eta_i)/x_i \quad i = 1, \ldots, n \ .$$

Denoting $\theta = n_\Sigma \mathbb{1} + n_\Delta \eta$, the problem then reads

$$\max_{\|\theta - n_\Sigma \mathbb{1}\|_\infty \leq n_\Delta} \theta^T (\mathbb{1} + \log(x)) - \theta^T \log(\theta) \ .$$

Its minimum is reached at

$$\theta^\star = \mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma \mathbb{1}, n_\Delta)}(x) \ ,$$

where $\mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma \mathbb{1}, n_\Delta)}$ is the projection on the infinity ball of radius n_Δ centered at $n_\Sigma \mathbb{1}$. We then get

$$\exp(-a_i^\star) = \mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma, n_\Delta)} \left(\exp[-(\tilde{Q} + \Lambda^\star)]_{i,\cdot}^T \exp(-c^\star) \right) / \left\{ \exp[-(\tilde{Q} + \Lambda^\star)]_{i,\cdot}^T \exp(-c^\star) \right\} \ .$$

Note that in the case where the clusters are constrained to all have the same size, i.e., $n_{\min} = n_{\max}$, the numerator reduces to n_Σ.

Minimization in c. Minimization in c is analogous and we get

$$\exp(-c^\star) = \mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma, n_\Delta)} \left(\exp[-(\tilde{Q} + \Lambda^\star)]_{\cdot,i}^T \exp(-a^\star)) \right) / \left\{ \exp[-(\tilde{Q} + \Lambda^\star)]_{\cdot,i}^T \exp(-a^\star) \right\} \ .$$

Define $u = \exp(-a)$, $v = \exp(-c)$, and $N = \exp(-(\tilde{Q} + \Lambda))$. Given the above equations, the steps of the alternating minimization are given by:

$$\begin{aligned}
N_{ij}^{(t+1)} &= m_{ij} / \left(u_i^{(t)} v_j^{(t)} \right) \ , && (i,j) \in \mathcal{K} \\
N_{ij}^{(t+1)} &= \exp(-\tilde{Q}_{ij}) \ , && (i,j) \notin \mathcal{K} \\
u_i^{(t+1)} &= \mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma, n_\Delta)} \left({N_{i,\cdot}^{(t+1)}}^T v^{(t)} \right) / \left({N_{i,\cdot}^{(t+1)}}^T v^{(t)} \right) , && i = 1, \ldots, n \\
v_i^{(t+1)} &= \mathrm{Proj}_{\mathcal{B}_\infty(n_\Sigma, n_\Delta)} \left({N_{\cdot,i}^{(t+1)}}^T u^{(t+1)} \right) / \left({N_{\cdot,i}^{(t+1)}}^T u^{(t+1)} \right) , && i = 1, \ldots, n \ .
\end{aligned}$$

Assuming these converge to values $N^\star$, $u^\star$, and $v^\star$, the final output is

$$M^\star = \mathrm{diag}(u^\star) N^\star \, \mathrm{diag}(v^\star) \ .$$

The algorithm is summarized in Algorithm 4.

B.2.2 Jacobian of the Sinkhorn iterations

Consider for $A \in \mathbb{R}^{n\times n}$ symmetric, the balancing problem

$$\begin{aligned} \min_{M\in\mathbb{R}^{n\times n}} \quad & \mathbf{Tr}(M^\top A) + \mu h(M) \\ \text{subject to} \quad & M\mathbb{1}_n = \alpha, M^\top \mathbb{1}_n = \beta \ , \end{aligned}$$

with $\alpha, \beta \in \mathbb{R}^n$, $h(M) = \sum_{ij} M_{ij}\log(M_{ij})$ and $\mathbb{1}_n$ denoting the vector of all ones in $\mathbb{R}^n$.

Its dual can be written

$$\min_{a,b\in\mathbb{R}^n} \exp(-a)^\top \exp(-\mu^{-1}A - \mathbb{1}_n\mathbb{1}_n^\top)\exp(-b) + a^\top\alpha + b^\top\beta \ ,$$

and after a change of variables

$$\min_{u,v} u^\top Q v - \log(u)^\top\alpha - \log(v)^\top\beta \ , \tag{B.3}$$

where $Q = \exp(-\mu^{-1}A - \mathbb{1}_n\mathbb{1}_n^\top)$.

Alternating minimization on (B.3) reads, starting from $v_0 = \mathbb{1}_n$, for $t \geq 0$,

$$u_{t+1} = \alpha./(Qv_t), \qquad v_{t+1} = \beta./(Q^\top u_{t+1}) \tag{B.4}$$

where ./ denotes the element-wise division of two vectors. The iterations (B.4) can be written in matrix form as

$$Q_{t+1} = \tilde{\Pi}_\beta(\Pi_\alpha(Q_t)) \ , \tag{B.5}$$

where for given vectors $\alpha, \beta \in \mathbb{R}^n$,

$$\Pi_\alpha(Q) = \mathrm{diag}(\alpha./(Q\mathbb{1}_n))Q \ , \qquad \tilde{\Pi}_\beta(Q) = Q\,\mathrm{diag}(\beta./(Q^\top\mathbb{1}_n)) = (\Pi_\beta(Q^\top))^\top \ ,$$

starting from $Q_0 = Q$ and we denoted for $x \in \mathbb{R}^n$, $\mathrm{diag}(x) = \sum_{i=1}^n x_i e_i e_i^\top$, with e_i the i^{th} canonical vector in $\mathbb{R}^n$. We present the computation of the Jacobian of the iterative process around a fixed point in the following proposition. This Jacobian drives the application of the generalized Ostrowski theorem presented by Soules (1991).

Proposition 65. *For a given Q^* such that $\Pi_\alpha(Q^*) = Q^*$ and $\Pi_\beta(Q^*) = Q^*$, the Jacobian of the vectorized composition $\pi = \mathrm{Vect} \circ \tilde{\Pi}_\beta \circ \Pi_\alpha \circ \mathrm{Vect}^{-1}$ at $q^* = \mathrm{Vect}(Q^*)$ reads*

$$\nabla \pi(q^*) = \sum_{i,j=1}^{n} (e_j e_j^\top - \mathbb{1}_n u_i^\top e_j e_j^\top) \otimes (e_i e_i^\top - e_i e_i^\top \mathbb{1}_n v_j^\top) \ ,$$

where $u_i = \frac{Q^\top e_i}{(Q \mathbb{1}_n)_i}$, $v_j = \frac{Q e_j}{(Q^\top \mathbb{1}_n)_j}$.

Proof. For a matrix $Q \in \mathbb{R}^{n \times n}$, denote by $\mathrm{Vect}(Q) \in \mathbb{R}^{n^2}$ the vectorized matrix (given by stacking the columns of Q). Denote by $\mathrm{Vect}^{-1} : \mathbb{R}^{n^2} \to \mathbb{R}^{n \times n}$ the inverse operation such that $\mathrm{Vect}^{-1}(\mathrm{Vect}(Q)) = Q$. Denote by T the linear operator in $\mathbb{R}^{n^2}$ such that for any $Q \in \mathbb{R}^{n \times n}$, $\mathrm{Vect}(Q^\top) = T \, \mathrm{Vect}(Q)$. We are interested in the vectorized formulation of Π_α, Π_β that read for $q \in \mathbb{R}^{n^2}$,

$$\pi_\alpha(q) = \mathrm{Vect}(\Pi_\alpha(\mathrm{Vect}^{-1}(q))) \qquad \tilde{\pi}_\beta(q) = T \pi_\beta(Tq)$$

Precisely, denote $\pi = \tilde{\pi}_\beta \circ \pi_\alpha$. The iterations (B.5) then read

$$q_{t+1} = \pi(q_t).$$

Our assumption is that π has a fixed point q^*. We need then to show that π is Frechet differentiable at q^* and that $\rho(\nabla \pi(q^*)) < 1$. Since

$$\nabla \pi(q) = \nabla \pi_\alpha(q) \nabla \tilde{\pi}_\beta(\pi_\alpha(q)) \ ,$$

it boils down to analyzing that $\nabla \pi_\alpha(q)$ and $\nabla \tilde{\pi}_\beta(\pi_\alpha(q))$ are defined on q^* and that $\rho(\nabla \pi_\alpha(q)) < 1$ and $\rho(\nabla \tilde{\pi}_\beta(\pi_\alpha(q))) < 1$.

We begin by computing $\nabla \pi_\alpha(q)$. Note that $\nabla \tilde{\pi}_\beta(q) = T^\top \nabla \pi_\beta(q) T^\top$. In the following denote by $A \otimes B$ the Kronecker product of two matrices A, B and I_n the identity matrix in $\mathbb{R}^{n \times n}$. Recall the Kronecker formula for matrices with appropriate sizes: $\mathrm{Vect}(AXB) = (B^\top \otimes A)\,\mathrm{Vect}(X)$. We have for $q \in \mathbb{R}^{n^2}$, denoting $Q = \mathrm{Vect}^{-1}(q)$, two formulations of π_α:

$$\pi_\alpha(q) = (\mathrm{I}_n \otimes \mathrm{diag}(\alpha./(Q\mathbb{1}_n)))q$$
$$\pi_\alpha(q) = (Q^\top \otimes \mathrm{I}_n)\,\mathrm{Vect}(\mathrm{diag}(\alpha./(Q\mathbb{1}_n))) = (Q^\top \otimes \mathrm{I}_n)E[\alpha./(\mathbb{1}_n^\top \otimes \mathrm{I}_n)q]$$

where $E = \sum_{i=1}^n (e_i \otimes e_i e_i^\top) \in \mathbb{R}^{n^2 \times n}$ such that $\mathrm{Vect}(\mathrm{diag}(x)) = Ex$ for $x \in \mathbb{R}^n$. We therefore get using the chain rule

$$\nabla \pi_\alpha(q) = \underbrace{(\mathrm{I}_n \otimes \mathrm{diag}(\alpha./(Q\mathbb{1}_n)))}_{A} + \underbrace{(\mathbb{1}_n \otimes \mathrm{I}_n)\,\mathrm{diag}(-\alpha./(Q\mathbb{1}_n).^2)E^\top (Q \otimes \mathrm{I}_n)}_{B},$$

where we used that $f : x \to \alpha./x$ for $x \in \mathbb{R}^n$ has gradient $\nabla f(x) = \mathrm{diag}(-\alpha./x.^2)$ with $.^2$ the element-wise application of the square operation and the gradient of $g : x \to Ax$ is $\nabla g(x) = A^\top$. Then we get

$$\begin{aligned}
A &= \sum_{i=1}^n (\mathrm{I}_n \otimes [\alpha_i/(Q\mathbb{1}_n)_i] e_i e_i^\top) = \sum_{i=1}^n [\alpha_i/(Q\mathbb{1}_n)_i](\mathrm{I}_n \otimes e_i e_i^\top)\,, \\
B &= -(\mathbb{1}_n \otimes \mathrm{I}_n)\left(\sum_{j=1}^n [\alpha_j/(Q\mathbb{1}_n)_j^2] e_j e_j^\top\right)\left(\sum_{i=1}^n (e_i^\top \otimes e_i e_i^\top)(Q \otimes \mathrm{I}_n)\right) \\
&= -\left(\sum_{j=1}^n [\alpha_j/(Q\mathbb{1}_n)_j^2](\mathbb{1}_n \otimes e_j e_j^\top)\right)\left(\sum_{i=1}^n (e_i^\top Q \otimes e_i e_i^\top)\right) \\
&= -\sum_{i=1}^n [\alpha_i/(Q\mathbb{1}_n)_i^2](\mathbb{1}_n (Q^\top e_i)^\top \otimes e_i e_i^\top)\,.
\end{aligned}$$

Therefore, denoting $u_i = \frac{Q^\top e_i}{(Q\mathbb{1}_n)_i}$, we have

$$\nabla \pi_\alpha(q) = \sum_{i=1}^n \frac{\alpha_i}{(Q\mathbb{1}_n)_i}\left(\left[\mathrm{I}_n - \mathbb{1}_n u_i^\top\right] \otimes e_i e_i^\top\right)\,.$$

We get similarly, denoting $v_j = \frac{Qe_j}{(Q^\top \mathbb{1}_n)_j}$,

$$\begin{aligned}\nabla \pi_\beta(q) &= \sum_{j=1}^{n} \frac{\beta_j}{(Q^\top \mathbb{1}_n)_j} T\left(\left[\mathrm{I}_n - \mathbb{1}_n v_j^\top\right] \otimes e_j e_j^\top\right) T \\ &= \sum_{j=1}^{n} \frac{\beta_j}{(Q^\top \mathbb{1}_n)_j} \left(e_j e_j^\top \otimes \left[\mathrm{I}_n - \mathbb{1}_n v_j^\top\right]\right) \ .\end{aligned}$$

Therefore, denoting $\tilde{q} = \pi_\alpha(q)$, $\tilde{Q} = \mathrm{Vect}^{-1}(\tilde{q})$ and $\tilde{v}_j = \frac{\tilde{Q}e_j}{(\tilde{Q}^\top \mathbb{1}_n)_j}$, we get

$$\nabla \pi(q) = \sum_{i,j=1}^{n} \frac{\alpha_i \beta_j}{(Q\mathbb{1}_n)_i (\tilde{Q}^\top \mathbb{1}_n)_j} (e_j e_j^\top - \mathbb{1}_n u_i^\top e_j e_j^\top) \otimes (e_i e_i^\top - e_i e_i^\top \mathbb{1}_n \tilde{v}_j^\top) \ .$$

In particular for q^* such that $\pi_\alpha(q^*) = q^*$ and $\pi_\beta(q^*) = q^*$, we get

$$\nabla \pi(q^*) = \sum_{i,j=1}^{n} (e_j e_j^\top - \mathbb{1}_n u_i^\top e_j e_j^\top) \otimes (e_i e_i^\top - e_i e_i^\top \mathbb{1}_n v_j^\top) \ .$$

□

B.3 An Alternative Relaxation

Bach and Harchaoui (2007) propose alternative relaxations of the labeling subproblem. Define $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ to be the eigenvalues of the equivalence matrix M and let $\lambda_0 \geq 0$. In Section 2.6 of their paper Bach and Harchaoui suggest solving the problem

$$\begin{aligned} \min_{M \in \mathbb{R}^{n \times n}} \quad & \mathbf{Tr}(M^T A) \\ \text{subject to} \quad & M = M^T \\ & \mathrm{trace}(M) = n \\ & M \succeq 0 \\ & \sum_{i=1}^{n} \min\left\{\frac{\lambda_i}{\lambda_0}, 1\right\} \geq k \end{aligned} \tag{B.6}$$

in the unsupervised setting. Note that the symmetric and positive semi-definite constraints imply that we can write $M = U\Lambda U^T$ where U contains an orthonormal set of eigenvectors of M and $\Lambda \geq 0$ is a diagonal matrix containing the corresponding eigenvalues. After rewriting $\mathbf{Tr}(M^T A) = \sum_{i=1}^n \lambda_i u_i^T A u_i$, with $u_i = U_{\cdot,i}$, we obtain the problem

$$\begin{aligned}
\min_{U \in \mathbb{R}^{n \times n}, \lambda_1, \ldots, \lambda_n \in \mathbb{R}} \quad & \sum_{i=1}^n \lambda_i u_i^T A u_i \\
\text{subject to} \quad & \sum_{i=1}^n \lambda_i = n \\
& \sum_{i=1}^n \min\left\{\frac{\lambda_i}{\lambda_0}, 1\right\} \geq k \\
& u_i^T u_i = 1 \quad \forall i \\
& u_i^T u_j = 0 \quad \forall i \neq j \\
& \lambda_i \geq 0 \quad \forall i \; .
\end{aligned}$$

Introducing Lagrange multipliers, we can rewrite the problem as

$$\begin{aligned}
\mathcal{L}(U, \Lambda, \alpha, \beta, \gamma, \delta, \epsilon) = & \sum_{i=1}^n \lambda_i u_i^T A u_i + \alpha \left(\sum_{i=1}^n \lambda_i - n \right) - \beta \left(\sum_{i=1}^n \min\left\{\frac{\lambda_i}{\lambda_0}, 1\right\} - k \right) \\
& + \sum_{i=1}^n \gamma_i (u_i^T u_i - 1) + \sum_{i \neq j} \delta_{ij} u_i^T u_j - \sum_{i=1}^n \epsilon_i \lambda_i \\
\text{subject to} \quad & \beta \geq 0, \quad \epsilon_i \geq 0 \quad \forall i \; ,
\end{aligned}$$

where $\alpha \in \mathbb{R}, \beta \in \mathbb{R}, \gamma \in \mathbb{R}^n, \delta \in \mathbb{R}^{n^2}, \epsilon \in \mathbb{R}^n$ and we define $\delta_{ii} = 0$ for all i. The optimal parameter values must satisfy the first order conditions

$$\begin{aligned}
& 2\lambda_i^\star A u_i^\star + 2\gamma_i^\star u_i^\star + \sum_{i \neq j} \delta_{ij}^\star u_j^\star = 0 \quad \forall i \\
& u_i^{\star T} A u_i^\star + \alpha^\star - \beta^\star \left[\frac{1}{2\lambda_0} (1 - \text{sign}(\lambda_i^\star - \lambda_0)) \right] - \epsilon_i^\star \ni 0 \quad \forall i \; .
\end{aligned} \tag{B.7}$$

From line B.7 we can see that UAU^T is diagonal, and hence U consists of a set of eigenvectors of A. Defining $0 \leq a_1 \leq a_2 \leq \cdots \leq a_n$ to be the eigenvalues of A, we can then rewrite the problem as

$$\begin{aligned} \min_{\lambda_1,\ldots,\lambda_n \in \mathbb{R}} \quad & \sum_{i=1}^{n} \lambda_i a_i && \text{(B.8)} \\ \text{subject to} \quad & \sum_{i=1}^{n} \lambda_i = n \\ & \sum_{i=1}^{n} \min\left\{\frac{\lambda_i}{\lambda_0}, 1\right\} \geq k && \text{(B.9)} \\ & \lambda_i \geq 0 \quad \forall i \ . \end{aligned}$$

To solve this problem, consider a possible solution $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n$. We will consider several cases. First, suppose there exists $i < j$ such that $\tilde{\lambda}_i, \tilde{\lambda}_j > \lambda_0$. Then define $\tilde{\lambda}'_i = \tilde{\lambda}_i + \tilde{\lambda}_j - \lambda_0$, $\tilde{\lambda}'_j = \lambda_0$, and $\tilde{\lambda}'_m = \tilde{\lambda}_m$ for $m \notin \{i, j\}$. Since $\tilde{\lambda}'_i, \tilde{\lambda}'_j \geq \lambda_0$ and $\tilde{\lambda}'_i + \tilde{\lambda}'_j = \tilde{\lambda}_i + \tilde{\lambda}_j$ the constraints are still satisfied. Therefore, since $a_i \leq a_j$, $\sum_{i=1}^n \tilde{\lambda}'_i a_i \leq \sum_{i=1}^n \tilde{\lambda}_i a_i$, and so we know that there always exists an optimum with at most one i such that $\lambda_i > \lambda_0$. Moreover, suppose that this index i is larger than 1. Then we could set $\tilde{\lambda}'_i = \tilde{\lambda}_1$, $\tilde{\lambda}'_1 = \tilde{\lambda}_i$, and $\tilde{\lambda}'_m = \tilde{\lambda}_m$ for $m \notin \{1, i\}$, thereby obtaining $\sum_{i=1}^n \tilde{\lambda}'_i a_i \leq \sum_{i=1}^n \tilde{\lambda}_i a_i$. Thus, there always exists an optimum $\lambda^\star_1, \ldots, \lambda^\star_n$ with $\lambda^\star_2, \ldots, \lambda^\star_n \leq \lambda_0$.

Next, suppose there exists $i < j$ such that $0 < \tilde{\lambda}_i, \tilde{\lambda}_j < \lambda_0$. Then define $\tilde{\lambda}'_i = \tilde{\lambda}_i + \min\{\lambda_0 - \tilde{\lambda}_i, \tilde{\lambda}_j\}$, $\tilde{\lambda}'_j = \tilde{\lambda}_j - \min\{\lambda_0 - \tilde{\lambda}_i, \tilde{\lambda}_j\}$, and $\tilde{\lambda}'_m = \tilde{\lambda}_m$ for $m \notin \{i, j\}$. Since $\tilde{\lambda}'_i, \tilde{\lambda}'_j \leq \lambda_0$ and $\tilde{\lambda}'_i + \tilde{\lambda}'_j = \tilde{\lambda}_i + \tilde{\lambda}_j$ the constraints are still satisfied. Therefore, since $a_i \leq a_j$, $\sum_{i=1}^n \tilde{\lambda}'_i a_i \leq \sum_{i=1}^n \tilde{\lambda}_i a_i$, we know that there always exists an optimum with at most one i such that $0 < \lambda_i < \lambda_0$. Now suppose that this i is not the largest index such that $\lambda_i > 0$. Then there exists an optimum with a $j > i$ such that $\lambda_j = \lambda_0$. Then we could set $\tilde{\lambda}'_i = \tilde{\lambda}_j$, $\tilde{\lambda}'_j = \tilde{\lambda}_i$, and $\tilde{\lambda}'_m = \tilde{\lambda}_m$ for $m \notin \{i, j\}$, thereby obtaining $\sum_{i=1}^n \tilde{\lambda}'_i a_i \leq \sum_{i=1}^n \tilde{\lambda}_i a_i$. Thus, there always exists an optimum $\lambda^\star_1, \ldots, \lambda^\star_n$ with $\lambda^\star_1 \geq \lambda_0$, $\lambda^\star_2, \ldots, \lambda^\star_{i-1} = \lambda_0$, $0 \leq \lambda^\star_i \leq \lambda_0$ for some i, and, if $i \neq n$, $\lambda^\star_{i+1}, \ldots, \lambda^\star_n = 0$.

Now from constraint (B.9) we can see that there must exist at least k non-zero λ_i's in the solution. If $n = k$, then we must have $\lambda_0 = 1$ and hence the optimum is given by $\lambda_1^\star, \ldots, \lambda_k^\star = 1$. Now consider the case where $n > k$. Suppose there exists a solution $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n$ such that $\tilde{\lambda}_{k+1} \neq 0$. Then since $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_k \geq \lambda_0$ we can set $\tilde{\lambda}'_1 = \tilde{\lambda}_1 + \tilde{\lambda}_{k+1}$, $\tilde{\lambda}_{k+1} = 0$, and $\tilde{\lambda}'_j = \tilde{\lambda}_j$ for $j \notin \{1, k+1\}$. This once again satisfies the constraints and $\sum_{i=1}^n \tilde{\lambda}'_i a_i \leq \sum_{i=1}^n \tilde{\lambda}_i a_i$. Therefore, there exists a solution such that $\lambda_1 \geq \lambda_0$ and $\lambda_2, \ldots, \lambda_k = \lambda_0$. In particular, a solution is $\lambda_1^\star = n - (k-1)\lambda_0$, $\lambda_2^\star, \ldots, \lambda_k^\star = \lambda_0$.

In summary, the optima of this problem depend on the values of k and n. In particular, we have:

- If $n > k$, an optimum is given by $\lambda_1^\star = n - (k-1)\lambda_0$, $\lambda_2^\star, \ldots, \lambda_k^\star = \lambda_0$, $\lambda_{k+1}^\star, \ldots, \lambda_n^\star = 0$.
- If $n = k$, the optimum is given by $\lambda_1^\star, \ldots, \lambda_k^\star = 1$.

Returning to the original problem (B.6), we therefore have that an optimal M is

$$M^\star = \sum_{i=1}^n \lambda_i^\star u_i u_i^T \ ,$$

where $u_1, \ldots, u_n$ are eigenvectors corresponding to the eigenvalues $a_1 \leq a_2, \cdots \leq a_n$ of A and where $\lambda_1^\star, \ldots, \lambda_n^\star$ are as defined above.

B.4 Additional Experimental Details

Here we provide additional details related to the datasets and the training.

B.4.1 Datasets

The details of the sizes and dimensions of each dataset we consider may be found in Table B.1. For the MAGIC dataset, which does not have a train/test split, we randomly split the data 75%/25% into train/test sets. For Gisette, MAGIC, and CIFAR-10 we set aside 20% of the training set to use as a validation set, while for MNIST we set aside the standard 17%. For

Dataset	Training size	Validation size	Test size	Dimension	# Classes
CIFAR-10	40,000	10,000	10,000	3,072	10
Gisette	4,800	1,200	1,000	5,000	2
MAGIC	8,026	2,006	4,755	10	2
MNIST	50,000	10,000	10,000	784	10

Table B.1: Details regarding the datasets used in the experiments

the unbalanced experiment we present on MNIST the size of the training dataset we use is always either 24,995 or 25,000, depending on rounding, with 50 labeled observations. Each class with labels 0-4 has the same number of unlabeled observations, and same for classes 5-9.

The datasets are transformed prior to usage as follows. Gisette is the scaled version found in the LibSVM database (Chang and Lin, 2011). MAGIC and MNIST are standardized. For CIFAR-10 we use the gradient map. As some of our experiments use the version of XSDC that assumes the classes are balanced, we randomly remove from the MAGIC dataset 4,223 observations in the training set with label 1.

B.4.2 XSDC with no labeled data

The XSDC algorithm for the purely unsupervised case is summarized in Algorithm 7. Aside from removing the supervised initialization step, the only difference lies in the estimation of $\hat{Y}_{\mathcal{U}}$ and the evaluation of the performance. Specifically, since we do not have any labeled data with which to perform nearest neighbor estimation, we instead use spectral clustering. Note that the cluster numbers output by spectral clustering do not necessarily map to the correct labels (e.g., cluster 0 might correspond to the label 1 rather than 0). Therefore, to evaluate the accuracy of the method we find the optimal relabeling of the classes that aligns with the true labels. We do so by solving a maximum weight matching problem with the Hungarian algorithm.

Algorithm 7 XSDC (when no labeled data is present)

Input: Unlabeled data $X_{\mathcal{U}}$
  Randomly initialized network parameters $V^{(1)}$
  Number of iterations T
for $t = 1, \ldots, T$ **do**
  $X^{(t)}, Y^{(t)} \leftarrow$ Draw minibatch of samples
  $M^{(t)} \leftarrow \text{MatrixBalancing}(A_\lambda(\Phi_{V^{(t)}}(X^{(t)})), Y^{(t)}Y^{(t)^T})$
  $V^{(t+1)} \leftarrow \text{ULR-SGO step}(\Phi_{V^{(t)}}(X^{(t)}), M^{(t)}, V^{(t)})$
end for
$\hat{Y}_{\mathcal{U}} \leftarrow \text{SpectralClustering}(M^{(T+1)})$
$\hat{W}, \hat{b} \leftarrow \text{RegLeastSquares}(X; \hat{Y}_{\mathcal{U}})$
Output: $\hat{Y}_{\mathcal{U}}, V^{(T)}, \hat{W}, \hat{b}$

B.4.3 Parameters

The algorithm proposed in this chapter and the models used require a large number of parameters to be set. Next we discuss the choices for these parameters.

Fixed parameters. The parameters that are fixed throughout the experiments and not validated are as follows. The number of filters in the networks is set to 32 and the network's parameters V are initialized layer-wise with 32 feature maps drawn uniformly at random from the output of the previous layer. The networks use the Nyström method to approximate the kernel at each layer. The regularization in the Nyström approximation is set to 0.001, and 20 Newton iterations are used to compute the inverse square root of the Gram matrix on the parameters V_ℓ at each layer ℓ, as in Chapter 2. The bandwidth is set to the median pairwise distance between the first 1000 observations for the single-layer networks. It is set to 0.6 for the convolutional networks. The batch size for both the labeled and unlabeled data is set to 4096 for Gisette and MAGIC and 1024 for MNIST and CIFAR-10 (due to GPU memory constraints). The features output by the network ϕ are centered and normalized so that on average they have unit ℓ_2 norm, as in Mairal et al. (2014b). The initial training phase on just the labeled data is performed for 100 iterations, as the validation loss has typically started

leveling off by 100 iterations. The entropic regularization parameter in the matrix balancing is set to the median absolute value of the entries in A. If this value results in divergence of the algorithm, it is multiplied by a factor of two until the algorithm converges. The value n_Δ is set to zero unless otherwise specified. The number of iterations of alternating minimization in the matrix balancing algorithm is set to 10. The number of nearest neighbors used for estimating the labels on the unlabeled data is set to 1.

Hold-out validation. Due to the large number of hyperparameters, we tune them sequentially as follows when labeled data, and hence a labeled validation set, exists. First, we tune the penalty λ on the classifier weights over the values 2^i for $i = -40, -39, \ldots, 0$. To do so, we train the classifier on only the labeled data using the initial random network parameters. We then re-validate this value every 100 iterations. Next, we tune the learning rate for the labeled data. For the modest values of $\alpha = \rho = 2^{-4}$ we validate the fixed learning rate for the labeled data over the values 2^i for $i = -10, -9, \ldots, 5$. To evaluate the performance the labels for the unlabeled data are estimated using 1-nearest neighbor. The labeled and unlabeled data are then used to train the classifier used to compute the performance. For the unbalanced experiments on MNIST only we then tune the minimum and maximum size of the classes over the values $0.01b, 0.02b, \ldots, 0.2b$, where b is the batch size (fixing the semi-supervised learning rate to 2^{-5}). For all other experiments we fix these values to b/k, where k is the number of classes in the dataset. We then tune the semi-supervised learning rate, again over the values 2^i for $i = -10, -9, \ldots, 5$. Following this, we validate ρ over the values 2^i for $i = -10, -9, \ldots, 10$. For the single-layer networks we then tune α over the values 2^i for $i = -10, -9, \ldots, 10$. For the convolutional networks we do not penalize the filters since they are constrained to lie on the sphere.

When no labeled data exists we consider the hyperparameters in the same manner as during the hold-out validation. First we consider the values 2^i for $i = -10, -9, \ldots, 5$ for the semi-supervised learning rate. Next we consider the values 2^i for $i = -40, -39, \ldots, 0$ for λ. We then consider the values 2^i for $i = -10, -9, \ldots, 10$ for ρ. Finally, if applicable, we consider

the values 2^i for $i = -10, -9, \ldots, 10$ for α. We report the best performance observed on the test set. Developing a method for tuning the hyperparameters on an unlabeled validation set is left for future work.

B.4.4 Training with competing labeling methods

In the comparisons we substitute our matrix balancing method with alternative labeling methods and retain the remainder of the XSDC algorithm. The pseudo-labeling code is our own, but we used code from Caron et al. (2018) to implement the k-means version of deep clustering.[1] Two important details regarding the implementations are as follows. First, for pseudo-labeling when some of the data is labeled we estimate W and b based on the labeled data in the current mini-batch, as that is what is done in XSDC. When labeled data is not present we estimate W and b based on the cluster assignments for the entire dataset. Second, for deep clustering we modify the dimension of the dimensionality reduction. In the original implementation the authors performed PCA, reducing the dimensionality of the features output by the network to 256. As the features output by the networks we consider have dimension less than 256, we instead keep the fewest number of components that account for 95% of the variance.

We perform the parameter tuning as follows. First, we follow the tuning procedure as detailed in Section B.4.3. For pseudo-labeling there are no additional parameters to tune. However, for deep clustering there are two additional parameters to tune: the number of clusters in k-means and the number of iterations between cluster updates. During the initial parameter tuning stage these parameters are set to the true number of clusters k, and 50 iterations, respectively. Afterward we tune these two remaining parameters sequentially. We first tune the number of clusters over the values $k, 2k, 4k, 8k, 16k, 32k$ where k is the true number of clusters. We then tune the number of iterations between cluster updates over the values $10, 25, 50, 100$.

---

[1] `https://github.com/facebookresearch/deepcluster`

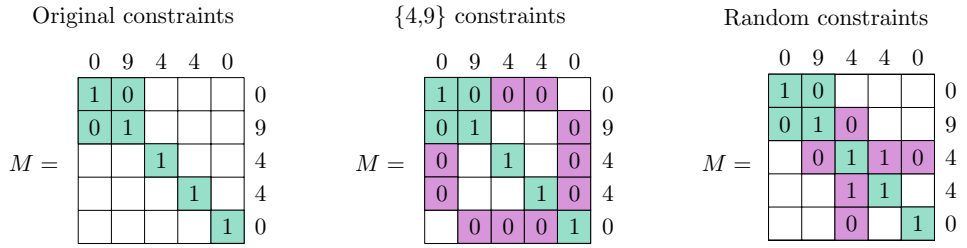


Figure B.1: Illustration of the kinds of additional constraints that were added. Green denotes the original constraints while purple denotes the constraints that were added. The numbers outside of the grids denote the true labels.

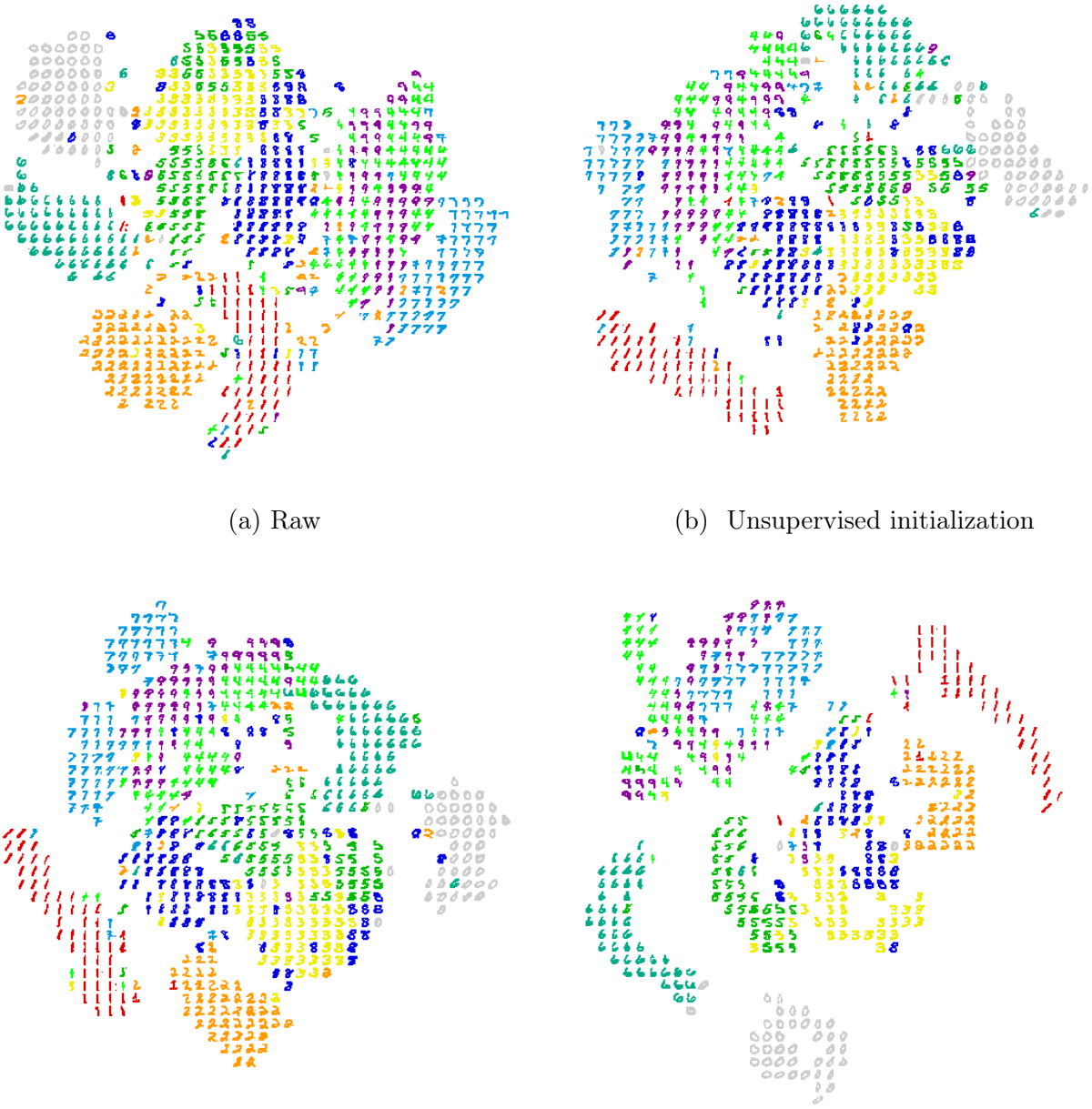
B.4.5 Additional constraints

In one set of experiments we examine the effect of adding additional constraints. We consider two types of constraints: (1) Constraints based on knowledge of whether the label was in the set $\{4, 9\}$ or not; and (2) Random correct must-link and must-not-link constraints among pairs of unlabeled observations and random correct must-not-link constraints between pairs of unlabeled and labeled observations.

The two types of constraints are illustrated in Figure B.1. Each grid point (i, j), if filled, denotes whether observations i and j have the same label (1) or not (0). The true labels are the values outside of the grids. Green backgrounds correspond to knowing the labels corresponding to (i, j). Purple backgrounds denote the additional known constraints. The left-most panel gives an example of an initial matrix M in which the labels corresponding to the first two observations are known (0 and 9). The second panel shows the entries we can fill in once we know whether each observation belongs to the set $\{4, 9\}$. Finally, the third panel shows random correct constraints. The constraint at entry $(2, 3)$ is a must-not-link constraint, whereas the constraint at entry $(3, 4)$ is a must-link constraint.

B.5 Additional Experimental Results

In this appendix we present additional experimental results that expand upon those presented in Section 3.5.



(a) Raw

(b) Unsupervised initialization

(c) Supervised initialization

(d) XSDC

Figure B.2: Visualizations of the unlabeled MNIST features obtained when training the LeNet-5 CKN with 50 labeled observations (where applicable). The CKN features were projected to 2-D using t-SNE. The features were obtained at different stages, as indicated in the sub-captions.

Visualization of feature representations. Figure B.2 visualizes the feature representations of unlabeled MNIST digits at different points in the training process. In each case the features were projected to 2-D with the Scikit-Learn (Pedregosa et al., 2011) implementation of t-SNE (Van Der Maaten and Hinton, 2008). Based on these t-SNE representations, the plots were then produced. The code to produce the plots was adapted from Andrej Karpathy's Matlab code.[2] It first scales the t-SNE representations so they all lie in $[0, 1]^2$ and creates an evenly-spaced 40×40 grid over $[0, 1]^2$. Then, for each square in the grid, it checks whether any image's t-SNE representation lies in that square. If any such images exist, it chooses one at random and displays the original image in that square. The images are then color-coded according to the ground-truth labels.

The batch size used to produce the t-SNE representations and the corresponding plots was 4096. The default parameters were used for generating the t-SNE representations. We created each plot with 20 random initializations and present the ones that visually appeared to be the best.

Figure B.2a visualizes the raw digits. By "raw" we mean that the digits were not input into the network; they were only standardized before applying t-SNE. Comparing this figure with Figure B.2b, which visualizes the feature representations from the network with randomly initialized filters, we can see that in both plots the 4's and 9's are mixed together. Furthermore, the 5's are split into two parts in Figure B.2a. Figure B.2c depicts the feature representations after the supervised initialization. Figure B.2c improves upon B.2b because the 4's and 9's are better-separated. However, the 5's have once again been split into two parts. Finally, Figure B.2d depicts the features representations after running XSDC. Comparing Figures B.2c and B.2d, we can see that XSDC tends to separate the clusters relative to the supervised initialization. While the 5's and 8's are generally all in one cluster after running XSDC, the 4's, 7's, and 9's are a bit less separated.

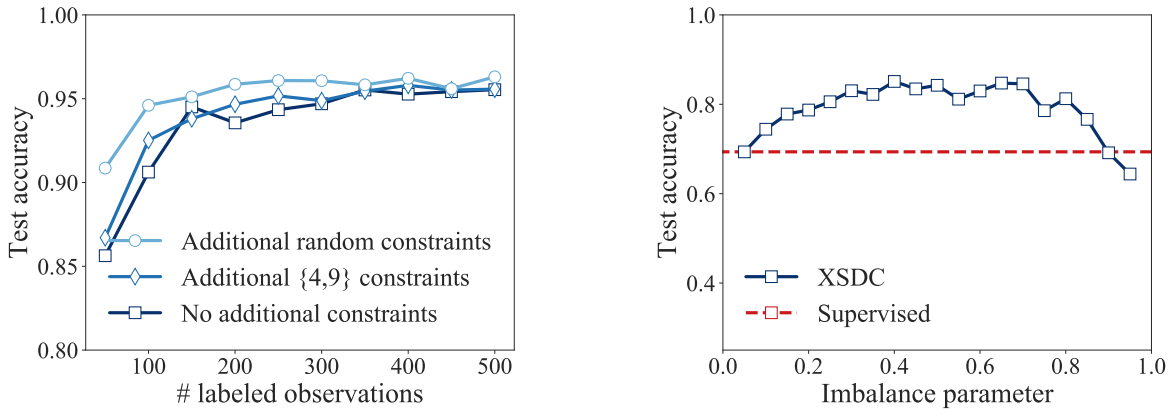
---

[2] `https://cs.stanford.edu/people/karpathy/cnnembed/`

Effect of additional constraints. Next, Figure B.3 compares the test accuracy of the LeNet-5 CKN on MNIST when adding additional constraints. As explained in Section 3.5, we consider two types of additional constraints. The first type of constraints are generated based on knowledge of whether each unlabeled point has a label in the set $\{4, 9\}$ or not. The second type of constraints is randomly generated constraints with approximately the same number of known entries in the adjacency matrix M as for the first type. We can see that both types of additional constraints generally improve the performance of the method. The largest gap occurs when there are 50 labeled observations, as expected. There the accuracies on MNIST when using the additional $\{4, 9\}$ constraints and the random constraints are 1% better and 5% better, respectively, than without them. This gap shrinks to 0.03% and 0.8% at 500 observations. As noted in Section 3.5, it is likely that the better performance with the random constraints than with the $\{4, 9\}$ constraints is because the former provide more information to be able to distinguish between labels for often-confused classes. Note that the performance with the $\{4, 9\}$ constraints at 150 labels is worse than without the constraints. This is probably because the hold-out validation overfit on the validation set.

Effect of unbalanced unlabeled data. Figure B.3b displays the accuracy of the LeNet-5 CKN on MNIST when 50 observations are labeled and the unlabeled data is unbalanced. To make the unlabeled data unbalanced we vary the fraction of the labels $\{0, 1, 2, 3, 4\}$ and the fraction of the labels $\{5, 6, 7, 8, 9\}$. Within each category 0-4 all of the labels are equally represented, and similarly for 5-9. The imbalance parameter in the plot denotes the fraction of the labels that are from the set $\{0, 1, 2, 3, 4\}$.

The performance is generally better when the data is closer to being balanced, as expected. The only time training with the unbalanced data is significantly worse than training on only the labeled data is when 95% of the unlabeled data is comprised of images of the digits 0-4. At that point the accuracy is 5% below the purely supervised performance.



(a) Effect of additional constraints

(b) Effect of unbalanced data

Figure B.3: Average accuracy across 10 trials of XSDC after training a LeNet-5 CKN on MNIST when adding additional constraints and varying the fraction of labeled data (left) and when varying the balance of the unlabeled data (right). The "additional {4,9} constraints" in (a) are derived from knowledge of whether the label for each unlabeled point lies in the set {4,9}. The imbalance parameter in (b) denotes the fraction of the labels that are from the set $\{0, 1, 2, 3, 4\}$. All classes in the set $\{0, 1, 2, 3, 4\}$ are equally represented, and similarly for $\{5, 6, 7, 8, 9\}$.

Sensitivity analysis. Next we perform a sensitivity analysis of the hyperparameters that are tuned via hold-out validation. To be consistent with the other experiments, the setting we choose is when training a LeNet-5 CKN on MNIST with 50 labels. For this architecture the parameters we validate over are the learning rates for the supervised initialization and semi-supervised learning and the penalties on the centered features and classifier weights. Here we vary one parameter at a time, fixing all others to their values obtained from hold-out validation (which were 2^{-7} and 2^{-6} for the learning rates of the supervised initialization and semi-supervised learning, respectively, and 2^{-5} for the penalty on the centered features. The penalty on the classifier weights was validated every 100 iterations.). We see that the most important parameter to tune is the semi-supervised learning rate. The other parameters only have a noticeable detrimental effect if they are too large.

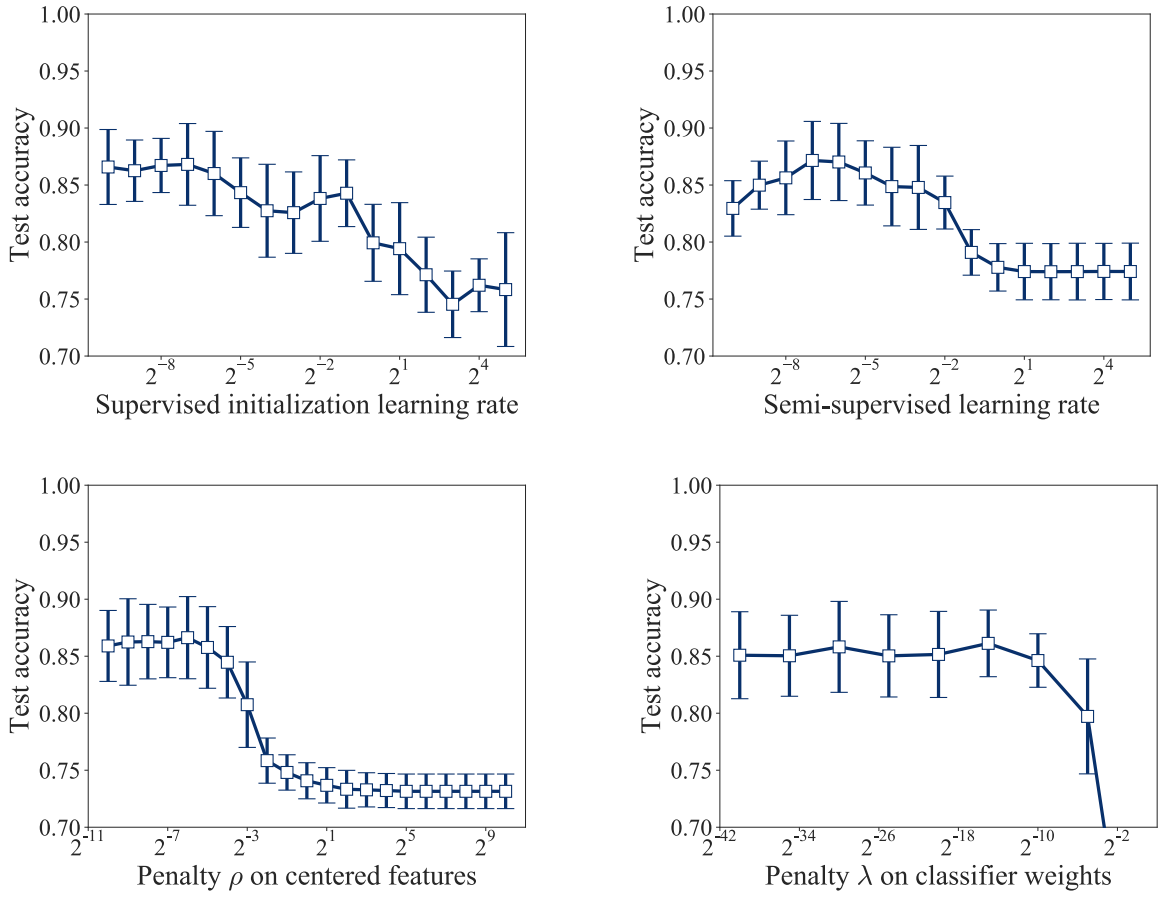
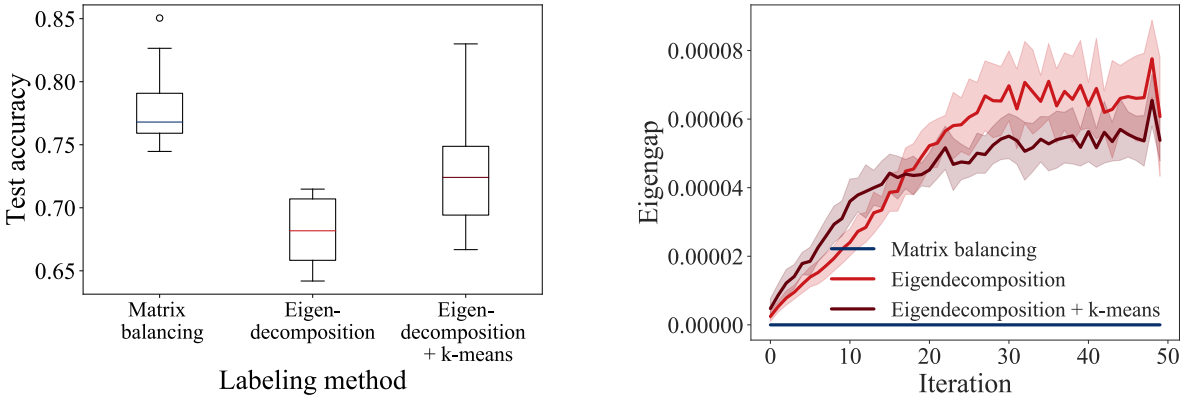


Figure B.4: Sensitivity analysis of the hyperparameters that are tuned with hold-out validation when using XSDC to train a LeNet-5 CKN on MNIST with 50 labeled observations.

Comparison to alternative relaxations. Finally, we compare the convex relaxation of the labeling suproblem presented in Section 3.4.2 to the relaxation proposed by Bach and Harchaoui (2007) presented in Appendix B.3. As accomodating constraints on cluster labels is less natural in the latter relaxation, we compare the relaxations when training a LeNet-5 CKN on MNIST with no labeled data. Figure B.5 compares our matrix balancing method, the eigendecomposition method from Appendix B.3, and the eigendecomposition method followed by k-means clustering. Prior to clustering, the rows of the eigenvector matrix were normalized to have unit ℓ_2 norm. The value of λ_0 was chosen from the set $\{0.01n_b, 0.02n_b, \ldots, 0.1n_b\}$,



(a) Effect of convex relaxation

(b) Evolution of the eigengap

Figure B.5: Performance of matrix balancing in comparison to eigendecomposition-based methods across 10 trials of training a LeNet-5 CKN on MNIST with no labeled data. The error bands in the plot on the right show one standard deviation from the mean.

where n_b is the size of a mini-batch, based on the performance on the validation set.

From Figure B.5a we can see that the convex relaxation used to derive the matrix balancing method is superior to the relaxations leading to the eigendecomposition-based methods. On average, matrix balancing performs 15% better than the eigendecomposition method and 7% better than the eigendecomposition method followed by k-means. This suggests that the constraint from the convex relaxation requiring the diagonal of M to consist of all 1's and/or the constraint requiring all entries of M to be positive are important for the performance of the labeling method.

In Figure B.5b we examine the eigengap of A across iterations. The eigengap is defined as $\lambda_{k+1} - \lambda_k$, where k is the number of classes and $\lambda_1 \leq \cdots \leq \lambda_n$ are the eigenvalues of A. As noted by Meila et al. (2005), having a larger eigengap makes the subspace spanned by the first k eigenvectors of A more stable to perturbations. From the figure we can see that the eigendecomposition-based methods tend to increase the eigengap as the learning proceeds. For the eigendecomposition method, the eigengap increased from 2×10^{-6} to 6×10^{-5} on average

after 50 iterations. Similarly, for the eigendecomposition method followed by k-means, the eigengap increased from 5×10^{-6} to 5×10^{-5} on average after 50 iterations. It is interesting to note that matrix balancing, which does not yield low-rank solutions $M^\star$, does not increase the eigengap across iterations. Nevertheless, it outperforms the eigendecomposition-based methods.

Appendix C

APPENDIX FOR CHAPTER 4

C.1 Convergence Analysis

The convergence of gradient descent on non-convex problems is governed by (1) the initial distance from the stationary point to which it converges; and (2) the Lipschitz constant of the gradient (Nesterov, 2004). In this appendix we prove conditions under which the objective is bounded below and under which the gradient of the objective with respect to w is Lipschitz. We then characterize the convergence of gradient descent in this setting.

We begin by proving that the objective is bounded below when the function ϕ as a function of its first argument is sufficiently smooth.

Proposition 11. *Assume $\phi(\cdot; w)$ is Lipschitz with constant $L_{\phi_x}(w)$ such that there exist $c_1, c_2 \in \mathbb{R}$ with $0 \leq L_{\phi_x}(w) \leq c_1 + c_2\|w\|_2^2$ for all w. Furthermore, assume the inputs $x_t \in \mathbb{R}^{d_x}$ lie in a bounded set with diameter D according to the Euclidean distance. Then for $\gamma \geq c_2 D\lambda$ the objective* (4.6) *is bounded below by $-\lambda(c_1 D + \sqrt{\epsilon})$.*

Proof. To show that the objective is bounded below it suffices to show that the sum of the penalties on the square root of the trace of the covariance matrix and on the squared norm of the parameters is bounded below. To this end, observe that for all i, t, t',

$$\begin{aligned}\left\|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\right\|_2^2 &\leq \max_{t,t'} \left\|\phi(x_t^{(i)}; w) - \phi(x_{t'}^{(i)}; w)\right\|_2^2 \\ &\leq L_{\phi_x}(w)^2 D^2 \ .\end{aligned}$$

Therefore,

$$\begin{aligned}\min_w \frac{1}{n}\sum_{i=1}^{n} \min_{\mathcal{T}\in\mathcal{C}_k^{(i)}(m^{(i)})} \mathcal{L}(\mathcal{T}, w; x^{(i)}) + \gamma\|w\|_2^2 &\geq -\lambda\sqrt{L_{\phi_x}(w)^2 D^2 + \epsilon} + \gamma\|w\|_2^2 \\ &\geq -\lambda L_{\phi_x}(w) D - \lambda\sqrt{\epsilon} + \gamma\|w\|_2^2 \\ &\geq -D\lambda(c_1 + c_2\|w\|_2^2) - \lambda\sqrt{\epsilon} + \gamma\|w\|_2^2 \\ &\geq -\lambda\sqrt{\epsilon} - c_1 D\lambda\,.\end{aligned}$$

□

Next we compute the gradient of the objective (4.6) with respect to w.

Proposition 12. *Assume that $\phi(x_t^{(i)};\cdot)$ is differentiable for all t, i. Then the gradient of the objective* (4.6)*, $\mathscr{L}(\{\mathcal{T}^{(i)}\}_{i=1}^n, w; \{x^{(i)}\}_{i=1}^n)$, with respect to w is given by*

$$\nabla_w \mathscr{L}(\{\mathcal{T}^{(i)}\}_{i=1}^n, w; \{x^{(i)}\}_{i=1}^n) = \frac{2}{n}\sum_{i=1}^{n}\frac{1}{T^{(i)}}\sum_{j=0}^{m^{(i)}}\sum_{t=t_j^{(i)}}^{t_{j+1}^{(i)}-1}\left(\nabla_w\phi(x_t^{(i)};w)\right)^T \tag{4.8}$$

$$\times\left\{\phi(x_t^{(i)};w) - \hat{\mu}_j^{(i)} - \frac{\lambda}{2}\left[\frac{1}{T^{(i)}}\sum_{t=1}^{T^{(i)}}\|\phi(x_t^{(i)};w) - \hat{\mu}^{(i)}\|_2^2 + \epsilon\right]^{-1/2}\left(\phi(x_t^{(i)};w) - \hat{\mu}^{(i)}\right)\right\} + 2\gamma w\,.$$

Proof. For a generic sequence x we will compute the partial derivatives of $\mathcal{L}$ with respect to the elements of the feature vector $\phi(x_t; w)$ with x_t in segment j. Define $\phi_t \coloneqq \phi(x_t; w)$ for all $t = 1, \ldots, T$. Observe that

$$\begin{aligned}&\frac{\partial}{\partial\phi_t}\left[\frac{1}{T}\sum_{j=0}^{m}\sum_{t=t_j}^{t_{j+1}-1}\|\phi_t - \hat{\mu}_j\|_2^2\right] \\ &= \frac{2}{T}\left(1 - \frac{1}{t_{j+1}-t_j}\right)(\phi_t - \hat{\mu}_j) - \frac{2}{(t_{j+1}-t_j)T}\sum_{t'\in\{t_j,\ldots,t_{j+1}-1\}\setminus\{t\}}(\phi_{t'} - \hat{\mu}_j) \\ &= \frac{2}{T}(\phi_t - \hat{\mu}_j)\end{aligned}$$

and, similarly,

$$\frac{\partial}{\partial \phi_t}\left[\frac{1}{T}\sum_{t=1}^{T}\|\phi_t-\hat{\mu}\|_2^2\right]=\frac{2}{T}\left(\phi_t-\hat{\mu}\right)\ .$$

The result then follows from the chain rule. □

Finally, we will compute the Lipschitz constant of the gradient of the objective with respect to w. To do so we will first compute the Lipschitz constant of $\mathcal{L}$ and the smoothness of $\mathcal{L}$ with respect to the learned features $\Phi^{(i)}=(\phi_1^{(i)},\ldots,\phi_{T^{(i)}}^{(i)})$ for $i=1,\ldots,n$. Afterward we will apply Lemma 68.

Lemma 66. *Assume there exists $M_\phi \geq 0$ such that that for all $x_t^{(i)}, w$, $\|\phi(x_t^{(i)};w)\|_2 \leq M_\phi$. Then*

$$\|\nabla_{\Phi^{(i)}}\mathcal{L}(\mathcal{T}^{(i)},w;x^{(i)})\|_2 \leq \frac{4}{\sqrt{T}}\left(1+\frac{\lambda}{2\sqrt{\epsilon}}\right)M_\phi\ .$$

Proof. For a generic sequence x define $\phi_t \coloneqq \phi(x_t;w)$ for all $t=1,\ldots,T$. Observe that

$$\begin{aligned}\|\nabla_{\phi_t}\mathcal{L}(\mathcal{T},w;x)\|_2 &= \frac{2}{T}\left\|(\phi_t-\hat{\mu}_j)-\frac{\lambda}{2}\left[\frac{1}{T}\sum_{t=1}^{T}\|\phi_t-\hat{\mu}\|_2^2+\epsilon\right]^{-1/2}(\phi_t-\hat{\mu})\right\|_2\\ &\leq \frac{4}{T}\left(1+\frac{\lambda}{2\sqrt{\epsilon}}\right)M_\phi\ .\end{aligned}$$

Hence,

$$\|\nabla_{\Phi^{(i)}}\mathcal{L}(\mathcal{T}^{(i)},w;x^{(i)})\|_2 \leq \frac{4}{\sqrt{T}}\left(1+\frac{\lambda}{2\sqrt{\epsilon}}\right)M_\phi\ .$$

□

Lemma 67. *Assume that for all x_t, w, $\|\phi(x_t;w)\|_2 \leq M_\phi$ for some M_ϕ. Then for each*

sequence i we have

$$\begin{aligned}
&\|\nabla_{\Phi^{(i)}}\mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) - \nabla_{\Phi^{(i)}}\mathcal{L}(\mathcal{T}^{(i)}, w'; x^{(i)})\|_F \\
&\leq \frac{1}{T}\left(4 + \frac{4\lambda}{\sqrt{\epsilon}} + \frac{64\lambda M_\phi^2}{\epsilon^{3/2}} + \frac{16\lambda^2 M_\phi^4}{\epsilon^3} + \frac{32\lambda^2 M_\phi^2}{\epsilon^2} + \frac{\lambda^2}{\epsilon}\right)^{1/2} \|\Phi^{(i)}(w) - \Phi^{(i)}(w')\|_F \,.
\end{aligned}$$

Proof. Let $\phi_t^{(i)}(w) \coloneqq \phi(x_t^{(i)}; w)$ for all $t = 1, \ldots, T^{(i)}$. Define

$$\begin{aligned}
f_t^{(i)}(w) &= \frac{2}{T^{(i)}}(\phi_t^{(i)}(w) - \hat{\mu}_j^{(i)}(w)) \,, \\
g_t^{(i)}(w) &= \frac{\lambda}{T^{(i)}}\left[\frac{1}{T^{(i)}}\sum_{t=1}^{T^{(i)}} \|\phi_t^{(i)}(w) - \hat{\mu}^{(i)}(w)\|_2^2 + \epsilon\right]^{-1/2} \left(\phi_t^{(i)}(w) - \hat{\mu}^{(i)}(w)\right) \,, \\
h^{(i)}(w) &= \frac{1}{T^{(i)}}\sum_{t=1}^{T^{(i)}} \|\phi_t^{(i)}(w) - \hat{\mu}^{(i)}(w)\|_2^2 + \epsilon \,,
\end{aligned}$$

and observe that

$$\begin{aligned}
&\|\nabla_{\Phi^{(i)}}\mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) - \nabla_{\Phi^{(i)}}\mathcal{L}(\mathcal{T}^{(i)}, w'; x^{(i)})\|_F^2 \\
&= \sum_{t=1}^{T^{(i)}} \left\| f_t^{(i)}(w) - g_t^{(i)}(w) - f_t^{(i)}(w') + g_t^{(i)}(w') \right\|_2^2 \\
&= \sum_{t=1}^{T^{(i)}} \left\{ \left\| f_t^{(i)}(w) - f_t^{(i)}(w') \right\|_2^2 - 2\left\langle f_t^{(i)}(w) - f_t^{(i)}(w'), g_t^{(i)}(w) - g_t^{(i)}(w') \right\rangle \right. \\
&\qquad \left. + \left\| g_t^{(i)}(w) - g_t^{(i)}(w') \right\|_2^2 \right\} \,.
\end{aligned}$$

We will bound each of the above terms in turn. Henceforth we will drop the superscripts denoting the sequence number for readability.

First note that we have

$$\sum_{t=t_j}^{t_{j+1}} \|f_t(w) - f_t(w')\|_2^2 = \sum_{t=t_j}^{t_{j+1}} \left\| \frac{2}{T}\left(\phi_t(w) - \hat{\mu}_j(w)\right) - \frac{2}{T}\left(\phi_t(w') - \hat{\mu}_j(w')\right) \right\|_2^2$$

$$
\begin{aligned}
&= \frac{4}{T^2} \sum_{t=t_j}^{t_{j+1}} \|(\phi_t(w) - \hat{\mu}_j(w)) - (\phi_t(w') - \hat{\mu}_j(w'))\|_2^2 \\
&= \frac{4}{T^2}(t_{j+1} - t_j) \operatorname{trace}\left[\widehat{\operatorname{Cov}}(\phi_t(w) - \phi_t(w'))\right] \\
&\leq \frac{4}{T^2}(t_{j+1} - t_j) \operatorname{trace}\left\{\widehat{\mathbb{E}}\left[((\phi_t(w) - \phi_t(w'))((\phi_t(w) - \phi_t(w'))^T\right]\right\} \\
&= \frac{4}{T^2} \sum_{t=t_j}^{t_{j+1}} \|\phi_t(w) - \phi_t(w')\|_2^2
\end{aligned}
$$

and hence

$$
\begin{aligned}
\sum_{t=1}^{T} \|f_t(w) - f_t(w')\|_2^2 &= \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}} \|f_t(w) - f_t(w')\|_2^2 \\
&\leq \frac{4}{T^2} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}} \|\phi_t(w) - \phi_t(w')\|_2^2 \\
&= \frac{4}{T^2} \sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2^2 .
\end{aligned}
$$

Next, we have

$$
\begin{aligned}
&\sum_{t=1}^{T} 2 \langle f_t(w) - f_t(w'), g_t(w) - g_t(w') \rangle \\
&= 2 \sum_{t=1}^{T} \left\langle f_t(w) - f_t(w'), \frac{\lambda}{T} h(w)^{-1/2}(\phi_t(w) - \hat{\mu}(w)) - \frac{\lambda}{T} h(w')^{-1/2}(\phi_t(w') - \hat{\mu}(w')) \right\rangle \\
&= \frac{2\lambda}{T} \sum_{t=1}^{T} \left\{ \left\langle f_t(w) - f_t(w'), h(w)^{-1/2}(\phi_t(w) - \hat{\mu}(w)) - h(w)^{-1/2}(\phi_t(w') - \hat{\mu}(w')) \right\rangle \right. \\
&\qquad \left. + \left\langle f_t(w) - f_t(w'), h(w)^{-1/2}(\phi_t(w') - \hat{\mu}(w')) - h(w')^{-1/2}(\phi_t(w') - \hat{\mu}(w')) \right\rangle \right\} \\
&= \frac{2\lambda}{T} \sum_{t=1}^{T} \left\{ \left\langle f_t(w) - f_t(w'), h(w)^{-1/2} \left\{(\phi_t(w) - \hat{\mu}(w)) - (\phi_t(w') - \hat{\mu}(w'))\right\} \right\rangle \right. \\
&\qquad \left. + \left\langle f_t(w) - f_t(w'), \left\{h(w)^{-1/2} - h(w')^{-1/2}\right\} (\phi_t(w') - \hat{\mu}(w')) \right\rangle \right\}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{2\lambda}{T}\sum_{t=1}^{T}\Bigg\{\frac{T}{2}h(w)^{-1/2}\left\|f_t(w)-f_t(w')\right\|_2^2 \\
&\qquad + \left\{h(w)^{-1/2}-h(w')^{-1/2}\right\}\left\langle f_t(w)-f_t(w'),\phi_t(w')-\hat{\mu}(w')\right\rangle\Bigg\} \\
&\leq \frac{2\lambda}{T}\sum_{t=1}^{T}\left\{\frac{2}{T\sqrt{\epsilon}}\left\|\phi_t(w)-\phi_t(w')\right\|_2^2 + 2M_\phi\left|h(w)^{-1/2}-h(w')^{-1/2}\right|\left\|f_t(w)-f_t(w')\right\|_2\right\} \\
&\leq \frac{4\lambda}{T^2\sqrt{\epsilon}}\sum_{t=1}^{T}\left\|\phi_t(w)-\phi_t(w')\right\|_2^2 + \frac{4\lambda M_\phi}{T}\left|h(w)^{-1/2}-h(w')^{-1/2}\right|\sum_{t=1}^{T}\left\|f_t(w)-f_t(w')\right\|_2 .
\end{aligned}
$$

We need to bound $|h(w)^{-1/2}-h(w')^{-1/2}|$ and $\sum_{t=1}^{T}\|f_t(w)-f_t(w')\|_2$ in terms of $\sum_{t=1}^{T}\|\phi_t(w)-\phi_t(w')\|_2^2$. To this end, observe that

$$
\begin{aligned}
&\left|h(w)^{-1/2}-h(w')^{-1/2}\right| \\
&= \left|\frac{\sqrt{h(w)}-\sqrt{h(w')}}{\sqrt{h(w)h(w')}}\right| \\
&\leq \frac{1}{\epsilon}\left|\sqrt{h(w)}-\sqrt{h(w')}\right| \\
&= \frac{1}{\epsilon}\frac{|h(w)-h(w')|}{\sqrt{h(w)}+\sqrt{h(w')}} \\
&\leq \frac{1}{2\epsilon^{3/2}}|h(w)-h(w')| \\
&= \frac{1}{2T\epsilon^{3/2}}\left|\sum_{t=1}^{T}\left(\left\|\phi_t(w)-\hat{\mu}(w)\right\|_2^2-\left\|\phi_t(w')-\hat{\mu}(w')\right\|_2^2\right)\right| \\
&\leq \frac{1}{2T\epsilon^{3/2}}\sum_{t=1}^{T}\left\|\phi_t(w)-\hat{\mu}(w)-\phi_t(w')+\hat{\mu}(w')\right\|_2\left(\left\|\phi_t(w)-\hat{\mu}(w)\right\|_2+\left\|\phi_t(w')-\hat{\mu}(w')\right\|_2\right) \\
&\leq \frac{2M_\phi}{T\epsilon^{3/2}}\sum_{t=1}^{T}\left\|\phi_t(w)-\hat{\mu}(w)-\phi_t(w')+\hat{\mu}(w')\right\|_2 \\
&\leq \frac{2M_\phi}{T\epsilon^{3/2}}\sum_{t=1}^{T}\left\{\left\|\phi_t(w)-\phi_t(w')\right\|_2+\left\|\hat{\mu}(w)-\hat{\mu}(w')\right\|_2\right\} \\
&= \frac{2M_\phi}{T\epsilon^{3/2}}\sum_{t=1}^{T}\left\{\left\|\phi_t(w)-\phi_t(w')\right\|_2+\left\|\frac{1}{T}\sum_{t=1}^{T}(\phi_t(w)-\phi_t(w'))\right\|_2\right\}
\end{aligned}
$$

$$\leq \frac{4M_\phi}{T\epsilon^{3/2}} \sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2$$

and

$$\begin{aligned}
\sum_{t=1}^{T} \|f_t(w) - f_t(w')\|_2 &= \frac{2}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}} \|(\phi_t(w) - \hat{\mu}_j(w)) - (\phi_t(w') - \hat{\mu}_j(w'))\|_2 \\
&\leq \frac{2}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}} \{\|\phi_t(w) - \phi_t(w')\|_2 + \|\hat{\mu}_j(w) - \hat{\mu}_j(w')\|_2\} \\
&\leq \frac{2}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}} \left\{ \|\phi_t(w) - \phi_t(w')\|_2 + \frac{1}{t_{j+1} - t_j} \sum_{t=t_j}^{t_{j+1}} \|\phi_t(w) - \phi_t(w')\|_2 \right\} \\
&\leq \frac{4}{T} \sum_{j=0}^{m} \sum_{t=t_j}^{t_{j+1}} \|\phi_t(w) - \phi_t(w')\|_2 \ .
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\sum_{t=1}^{T} 2 \left\langle f_t(w) - f_t(w'), g_t(w) - g_t(w') \right\rangle \\
&\leq \frac{4\lambda}{T^2\sqrt{\epsilon}} \sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2^2 + \frac{4\lambda M_\phi}{T} \left(\frac{4M_\phi}{T\epsilon^{3/2}} \right) \left(\frac{4}{T} \right) \left(\sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2 \right)^2 \\
&= \frac{4\lambda}{T^2\sqrt{\epsilon}} \sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2^2 + \frac{64\lambda M_\phi^2}{T^3\epsilon^{3/2}} \left(\sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2 \right)^2 \\
&\leq \frac{4\lambda}{T^2\sqrt{\epsilon}} \sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2^2 + \frac{64\lambda M_\phi^2}{T^2\epsilon^{3/2}} \sum_{t=1}^{T} \|\phi_t(w) - \phi_t(w')\|_2^2 \ .
\end{aligned}$$

Finally, we have

$$\begin{aligned}
&\sum_{t=1}^{T} \|g_t(w) - g_t(w')\|_2^2 \\
&= \sum_{t=1}^{T} \left\| \frac{\lambda}{T} h(w)^{-1/2} \left(\phi_t(w) - \hat{\mu}(w)\right) - \frac{\lambda}{T} h(w')^{-1/2} \left(\phi_t(w') - \hat{\mu}(w')\right) \right\|_2^2
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=1}^{T} \frac{\lambda^2}{T^2} \Big\| (h(w)^{-1/2} - h(w')^{-1/2}) \left(\phi_t(w) - \hat{\mu}(w)\right) \\
&\qquad\qquad + h(w')^{-1/2} \left[(\phi_t(w) - \hat{\mu}(w)) - (\phi_t(w') - \hat{\mu}(w')) \right] \Big\|_2^2 \\
&= \sum_{t=1}^{T} \frac{\lambda^2}{T^2} \Big\{ \Big\| (h(w)^{-1/2} - h(w')^{-1/2}) \left(\phi_t(w) - \hat{\mu}(w)\right) \Big\|_2^2 \\
&\quad + 2 \Big\langle \left(h(w)^{-1/2} - h(w')^{-1/2}\right) \left(\phi_t(w) - \hat{\mu}(w)\right), \\
&\qquad h(w')^{-1/2} \left[(\phi_t(w) - \hat{\mu}(w)) - (\phi_t(w') - \hat{\mu}(w')) \right] \Big\rangle \\
&\quad + \Big\| h(w')^{-1/2} \left[(\phi_t(w) - \hat{\mu}(w)) - (\phi_t(w') - \hat{\mu}(w')) \right] \Big\|_2^2 \Big\} \\
&\leq \frac{\lambda^2}{T^2} \Big\{ 4T M_\phi^2 \left| (h(w)^{-1/2} - h(w')^{-1/2}) \right|^2 \\
&\quad + \frac{4M_\phi}{\sqrt{\epsilon}} \left| h(w)^{-1/2} - h(w')^{-1/2} \right| \sum_{t=1}^{T} \left\| (\phi_t(w) - \hat{\mu}(w)) - (\phi_t(w') - \hat{\mu}(w')) \right\|_2 \\
&\quad + \frac{1}{\epsilon} \left\| (\phi_t(w) - \hat{\mu}(w)) - (\phi_t(w') - \hat{\mu}(w')) \right\|_2^2 \Big\} \\
&\leq \frac{4\lambda^2 M_\phi^2}{T} \left(\frac{2M_\phi}{T\epsilon^{3/2}} \sum_{t=1}^{T} \left\| \phi_t(w) - \hat{\mu}(w) - \phi_t(w') + \hat{\mu}(w') \right\|_2 \right)^2 \\
&\quad + \frac{4\lambda^2 M_\phi}{T^2\sqrt{\epsilon}} \left(\frac{4M_\phi}{T\epsilon^{3/2}} \sum_{t=1}^{T} \left\| \phi_t(w) - \phi_t(w') \right\|_2 \right) \left(2 \sum_{t=1}^{T} \left\| \phi_t(w) - \phi_t(w') \right\|_2 \right) \\
&\quad + \frac{\lambda^2}{T^2\epsilon} \sum_{t=1}^{T} \left\| \phi_t(w) - \phi_t(w') \right\|_2^2 \\
&\leq \frac{16\lambda^2 M_\phi^4}{T^2\epsilon^3} \sum_{t=1}^{T} \left\| \phi_t(w) - \hat{\mu}(w) - \phi_t(w') + \hat{\mu}(w') \right\|_2^2 + \left(\frac{32\lambda^2 M_\phi^2}{T^2\epsilon^2} + \frac{\lambda^2}{T^2\epsilon} \right) \sum_{t=1}^{T} \left\| \phi_t(w) - \phi_t(w') \right\|_2^2 \\
&\leq \left(\frac{16\lambda^2 M_\phi^4}{T^2\epsilon^3} + \frac{32\lambda^2 M_\phi^2}{T^2\epsilon^2} + \frac{\lambda^2}{T^2\epsilon} \right) \sum_{t=1}^{T} \left\| \phi_t(w) - \phi_t(w') \right\|_2^2 .
\end{aligned}
$$

Therefore, we have

$$
\begin{aligned}
&\left\| \nabla_{\Phi^{(i)}} \mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) - \nabla_{\Phi^{(i)}} \mathcal{L}(\mathcal{T}^{(i)}, w'; x^{(i)}) \right\|_F^2 \\
&\leq \frac{1}{T^2} \left(4 + \frac{4\lambda}{\sqrt{\epsilon}} + \frac{64\lambda M_\phi^2}{\epsilon^{3/2}} + \frac{16\lambda^2 M_\phi^4}{\epsilon^3} + \frac{32\lambda^2 M_\phi^2}{\epsilon^2} + \frac{\lambda^2}{\epsilon} \right) \sum_{t=1}^{T} \left\| \phi_t(w) - \phi_t(w') \right\|_2^2 .
\end{aligned}
$$

□

Lemma 68. *Let $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1 \times d_2}$ and $g : \mathbb{R}^{d_0} \to \mathbb{R}^{d_2}$. Fix a norm $\|\cdot\|$. Assume that f and g are bounded, i.e., there exist M_f and M_g such that $\|f(x)\| \leq M_f$ and $\|g(x)\| \leq M_g$ for all $x \in \mathbb{R}^{d_0}$. Moreover, assume that f is Lipschitz with constant L_f and g is Lipschitz with constant L_g. Then the function $h(x) = f(x)g(x)$ is Lipschitz with constant $L_h \coloneqq M_f L_g + M_g L_f$.*

Proof. Observe that for all $x, y \in \mathbb{R}^{d_0}$,

$$\begin{aligned}
\|h(x) - h(y)\| &= \|f(x)g(x) - f(y)g(y)\| \\
&= \|f(x)g(x) - f(x)g(y) + f(x)g(y) - f(y)g(y)\| \\
&\leq \|f(x)g(x) - f(x)g(y)\| + \|f(x)g(y) - f(y)g(y)\| \\
&\leq \|f(x)\|\|g(x) - g(y)\| + \|g(y)\|\|f(x) - f(y)\| \\
&\leq M_f L_g \|x - y\| + M_g L_f \|x - y\| \\
&= (M_f L_g + M_g L_f)\|x - y\| \ .
\end{aligned}$$

□

Based on the previous two lemmas we may now obtain the following result.

Lemma 69. *Define $\phi_t^{(i)}(w) = \phi(x_t^{(i)}; w)$ for all t, i. Assume $\phi_t^{(i)}(\cdot)$ is differentiable and there exist constants $M_\phi, M_{\nabla_w \phi} \geq 0$ such that for all $x_t^{(i)}$ and w we have $\|\phi_t^{(i)}(w)\|_2 \leq M_\phi$ and $\|\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)}(w))\|_F \leq M_{\nabla_w \phi}$. Moreover, assume there exists a constant $L_{\nabla_w \phi} \geq 0$ such that for all t and i, $\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)})$ is Lipschitz with constant $L_{\nabla_w \phi}$. Then*

$$\begin{aligned}
&\|\nabla_w \mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) - \nabla_w \mathcal{L}(\mathcal{T}^{(i)}, w'; x^{(i)})\|_F \\
&\leq \left\{ \frac{4}{\sqrt{T^{(i)}}} \left(1 + \frac{\lambda}{2\sqrt{\epsilon}}\right) M_\phi L_{\nabla_w \phi} + \frac{M_{\nabla_w \phi}}{\sqrt{T^{(i)}}} \sqrt{4 + \frac{4\lambda}{\sqrt{\epsilon}} + \frac{64\lambda M_\phi^2}{\epsilon^{3/2}} + \frac{16\lambda^2 M_\phi^4}{\epsilon^3} + \frac{32\lambda^2 M_\phi^2}{\epsilon^2} + \frac{\lambda^2}{\epsilon}} \right\} \\
&\quad \times \|w - w'\|_F \ .
\end{aligned}$$

Proof. Observe that

$$\nabla_w \mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) = \left(\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\Phi^{(i)})\right)^T \nabla_{\mathrm{vec}(\Phi^{(i)})} \mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)}) \ .$$

Note that by assumption $\|\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\Phi^{(i)})\|_F \leq M_{\nabla_w \phi} \sqrt{T^{(i)}}$ and $\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\Phi^{(i)})$ is Lipschitz with constant $L_{\nabla_w \phi}$. The result then follows from Lemmas 66-68. □

From Lemma 69 we can obtain a bound on the Lipschitz constant of the objective $\mathscr{L}$.

Proposition 13. *Define $\phi_t^{(i)}(w) = \phi(x_t^{(i)}; w)$ for all t, i. Assume $\phi_t^{(i)}(\cdot)$ is differentiable and there exist constants $M_\phi, M_{\nabla_w \phi} \geq 0$ such that for all $x_t^{(i)}$ and w we have $\|\phi_t^{(i)}(w)\|_2 \leq M_\phi$ and $\|\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)}(w))\|_F \leq M_{\nabla_w \phi}$. Moreover, assume there exists a constant $L_{\nabla_w \phi} \geq 0$ such that for all t and i, $\nabla_{\mathrm{vec}(w)} \mathrm{vec}(\phi_t^{(i)})$ is Lipschitz with constant $L_{\nabla_w \phi}$. Then the gradient of the objective $\mathscr{L}$ is Lipschitz with constant*

$$L_{\nabla \mathscr{L}} \coloneqq \frac{4 M_\phi L_{\nabla_w \phi}}{\sqrt{T_{\min}}} \left(1 + \frac{\lambda}{2\sqrt{\epsilon}}\right) + \frac{M_{\nabla_w \phi}}{\sqrt{T_{\min}}} \sqrt{4 + \frac{4\lambda}{\epsilon^{1/2}} + \frac{64 \lambda M_\phi^2}{\epsilon^{3/2}} + \frac{\lambda^2}{\epsilon} + \frac{32 \lambda^2 M_\phi^2}{\epsilon^2} + \frac{16 \lambda^2 M_\phi^4}{\epsilon^3}} + 2\gamma \ ,$$

where $T_{\min}$ is the length of the shortest sequence.

Proof. The result follows from the fact that an average of n Lipschitz functions all with Lipschitz constant upper bounded by L is also Lipschitz with constant L and the fact that the smoothness constant of the function $\gamma \|w\|_F^2$ is 2γ. □

From Proposition 13 we can obtain a bound on the convergence rate of gradient descent with a constant step size. This is stated in Proposition 14 in Section 4.5.

C.2 Infimal Convolution Smoothing

As noted in Section 4.5.2, the idea behind smoothing is to solve a sequence of smooth problems that are easier to solve than the original problem. In this appendix we derive the objective

function and gradient that result from smoothing the min function over the change-point locations in Problem (4.6).

We will compute the smoothing by rewriting the inner portion of the objective $\mathscr{L}$ in terms of a max function $g(z) = \max\{z_1, \ldots, z_p\}$ rather than a min function. The Moreau proximal approximation of g (Moreau, 1965) leads to the smoothed function

$$\begin{aligned} g_\alpha(x) &= \inf_{u \in \mathbb{R}^p} \left\{ g(u) + \frac{1}{2\alpha} \|u - x\|_2^2 \right\} \\ &= \max_{u \in \mathbb{R}^p} \left\{ \langle u, x \rangle - g^*(u) - \frac{\alpha}{2} \|u\|_2^2 \right\} , \end{aligned}$$

where $g^*(u) \coloneqq \sup_{x \in \mathbb{R}^p} \{\langle x, u \rangle - g(x)\}$ is the convex conjugate of g and the latter equality comes from the Fenchel duality theorem (Rockafellar, 1970). This smoothing can be viewed as applying the infimal-convolution of g with the function $q(x) = 0.5\|u\|_2^2/\alpha$. Rather than using this quadratic function, one could instead use a different convex and continuously differentiable function $q : \mathbb{R}^p \to \mathbb{R}$ with Lipschitz gradient (Beck and Teboulle, 2012). We will use the ℓ_2^2 smoothing of g given by $\ell_2^2(u) = 0.5(\|u\|_2^2 - 1)$ so that the smoothed max function will always be larger than the max function itself. The next result, which was stated without proof by Pillutla et al. (2019), computes the smoothed function in closed form.

Proposition 70. *Define the function $g : \mathbb{R}^p \to \mathbb{R}$ as $g(z) = \max_{m=1,\ldots,p} z_m$ (which is convex but non-smooth) and the smoothing function $\ell_2^2 : \mathbb{R}^p \to \mathbb{R}$ as $\ell_2^2(u) = 0.5(\|u\|_2^2 - 1)$. Then smoothing g by $\alpha\ell_2^2$ gives*

$$\begin{aligned} g_\alpha(z) &\coloneqq \max_{u \in \mathbb{R}^p} \left\{ \langle u, z \rangle - g^*(u) - \alpha \ell_2^2(u) \right\} \\ &= \langle z, \operatorname{proj}_{\Delta^{p-1}}(z/\alpha) \rangle - \frac{\alpha}{2} \| \operatorname{proj}_{\Delta^{p-1}}(z/\alpha) \|^2 + \frac{\alpha}{2} , \end{aligned}$$

where g^* is the convex conjugate of g and Δ^{p-1} is the $(p-1)$-dimensional probability simplex, i.e., the set of vectors $x \in \mathbb{R}^p$ satisfying $x_i \geq 0$ for all $i = 1, \ldots, p$ with $\sum_{i=1}^p x_i = 1$.

To prove this result we will make use of the following lemma.

Lemma 71. *The convex conjugate of* $g(z) \coloneqq \max_{m=1,\ldots,p} z_m$ *is given by*

$$\begin{aligned} g^*(z) &\coloneqq \sup_{u \in \mathbb{R}^p} \left\{ \langle z, u \rangle - \max_{m=1,\ldots,p} u_m \right\} \\ &= \begin{cases} 0, & z \in \Delta^{p-1} \\ \infty, & \textit{else} \, . \end{cases} \end{aligned}$$

Proof. We will prove this result by considering the following four cases:

- Case 1: There exists z_m with $m \in \{1, \ldots, p\}$ such that $z_m < 0$.
  Setting $u = -Me_m$ where e_m is the mth unit vector and taking $M \to \infty$ shows that in this case $g^*(z) = \infty$.

- Case 2: $z_m \geq 0$ for all $m = 1, \ldots, p$ and $\sum_{m=1}^p z_m > 1$.
  Observe that

$$\begin{aligned} \langle z, u \rangle - \max_{m=1,\ldots,p} u_m &= \sum_{m=1}^p u_m z_m - \max_{m=1,\ldots,p} u_m \\ &\geq \left(\min_{m=1,\ldots,p} u_m \right) \sum_{m=1}^p z_m - \max_{m=1,\ldots,p} u_m \, . \end{aligned} \tag{C.1}$$

  Setting $u = M\mathbb{1}_p$ where $\mathbb{1}_p$ is the p-dimensional vector of all 1's and taking $M \to \infty$ shows that in this case $g^*(z) = \infty$.

- Case 3: $z_m \geq 0$ for all $m = 1, \ldots, p$ and $\sum_{m=1}^p z_m < 1$.
  By inequality (C.1), setting $u = -M\mathbb{1}_p$ and taking $M \to \infty$ shows that in this case $g^*(z) = \infty$.

- Case 4: $z_m \geq 0$ for all $m = 1, \ldots, p$ and $\sum_{m=1}^p z_m = 1$.
  Observe that

$$\langle z, u \rangle - \max_{m=1,\ldots,p} u_m = \sum_{m=1}^p u_m z_m - \max_{m=1,\ldots,p} u_m$$

$$\leq \left(\max_{m=1,\dots,p} u_m\right) \sum_{m=1}^{p} z_m - \max_{m=1,\dots,p} u_m$$
$$= 0 \,,$$

and this upper bound is achievable with $u = 0$. Therefore, in this case, $g^*(z) = 0$.

□

Now we return to the proof of Proposition 70.

Proof of Proposition 70. By Lemma 71 we have that

$$\begin{aligned} g_\alpha(z) &\coloneqq \max_{u \in \mathbb{R}^p} \left\{ \langle u, z \rangle - g^*(u) - \alpha \ell_2^2(u) \right\} \\ &= \max_{u \in \Delta^{p-1}} \left\{ \langle u, z \rangle - \frac{\alpha}{2} \|u\|_2^2 \right\} + \frac{\alpha}{2} \\ &= \max_{u \in \Delta^{p-1}} -\frac{\alpha}{2} \left\| u - \frac{z}{\alpha} \right\|_2^2 + \frac{\alpha}{2} \left\| \frac{z}{\alpha} \right\|_2^2 + \frac{\alpha}{2} \\ &= \langle z, \operatorname{proj}_{\Delta^{p-1}}(z/\alpha) \rangle - \frac{\alpha}{2} \| \operatorname{proj}_{\Delta^{p-1}}(z/\alpha) \|^2 + \frac{\alpha}{2} \,. \end{aligned}$$

□

Now, noting that $\max_{m=1,\dots,p} z_m = -\min_{m=1,\dots,p} -z_m$, we can obtain the smoothing for the objective $\mathscr{L}$.

Corollary 72. *Define the function $h : \mathbb{R}^p \to \mathbb{R}$ as $h(z) = \min_{m=1,\dots,p} z_m$ (which is concave but non-smooth) and the smoothing function $\ell_2^2 : \mathbb{R}^p \to \mathbb{R}$ as $\ell_2^2(u) = 0.5(\|u\|_2^2 - 1)$. Then smoothing h by $\alpha\ell_2^2$ in (4.6) gives*

$$\min_w \frac{1}{n} \sum_{i=1}^{n} h_\alpha(\mathcal{L}(w; x^{(i)})) + \gamma \|w\|_2^2 \,,$$

where $h_\alpha(z) \coloneqq \langle z, \operatorname{proj}_{\Delta^{p-1}}(-z/\alpha) \rangle + \frac{\alpha}{2} \| \operatorname{proj}_{\Delta^{p-1}}(-z/\alpha) \|^2 - \frac{\alpha}{2}$.

Note that smoothing $\min_{\mathcal{T}^{(i)} \in \mathcal{C}^{(i)}(m^{(i)})} \mathcal{L}(\mathcal{T}^{(i)}, w; x^{(i)})$ is the same as smoothing the function $\min_{\mathcal{T}^{(i)} \in \mathcal{C}^{(i)}(m^{(i)})} \frac{1}{T^{(i)}} \sum_{j=0}^{m^{(i)}} \sum_{t=t_j^{(i)}}^{t_{j+1}^{(i)}-1} \left\| \phi(x_t^{(i)}; w) - \hat{\mu}_j^{(i)} \right\|_2^2$ and then subtracting the penalty term on the square root of the trace of the covariance matrix, $\lambda \sqrt{\frac{1}{T^{(i)}} \sum_{t=1}^{T^{(i)}} \|\phi(x_t^{(i)}; w) - \hat{\mu}^{(i)}\|_2^2 + \epsilon}$. This can be shown using Lemma 3.1.4 of Nesterov (2004).

C.3 Perturbation Analysis

In this appendix we bound the optimization error introduced when minimizing the objective $\mathscr{L}$. In particular, we provide upper and lower bounds on the Frobenius distance between segmentations obtained with the estimated $\hat{w}$ found by optimizing $\mathscr{L}$ and with a global optimum $w^\star$. In this section we will write the objective function (4.1) as $\|x - \Pi_{\mathcal{T}} x\|_F^2 / T$, as done by Arlot et al. (2019), where $\Pi_{\mathcal{T}}$ is the normalized equivalence matrix defined in Section 4.5.3.

Our upper bound, stated next, requires assuming that the eigenvalues of the Gram matrix are bounded below. The proof largely follows that of Theorem 11.1 of Hastie et al. (2015).

Proposition 73. *Let $\mathcal{T}$ and $\mathcal{T}'$ be two segmentations of a sequence x with feature map ϕ parameterized by w such that $\mathcal{L}(\mathcal{T}, w; x) \leq \mathcal{L}(\mathcal{T}', w; x)$. Assume that the smallest eigenvalue of the normalized Gram matrix $\Phi(x; w)\Phi(x; w)^T / T$, $\lambda_{\min}$, satisfies $\lambda_{\min} > 0$. Then an upper bound on the Frobenius norm of the difference in the segment assignment matrices is given by*

$$d_F(\mathcal{T}, \mathcal{T}') \leq \frac{2}{\lambda_{\min} T} \|\Phi(x; w)\|_F \|\Phi(x; w) - \Pi_{\mathcal{T}'} \Phi(x; w)\|_F \, .$$

Proof. If $\mathcal{T} = \mathcal{T}'$ then the result is trivial. Therefore, consider the case where $\mathcal{T} \neq \mathcal{T}'$. Since $\mathcal{L}(\mathcal{T}, w; x) \leq \mathcal{L}(\mathcal{T}', w; x)$ by assumption, we have

$$\|\Phi(x; w) - \Pi_{\mathcal{T}} \Phi(x; w)\|_F^2 \leq \|\Phi(x; w) - \Pi_{\mathcal{T}'} \Phi(x; w)\|_F^2 \, .$$

Now let $d = \Pi_{\mathcal{T}} - \Pi_{\mathcal{T}'}$ and $e = \Phi(x;w) - \Pi_{\mathcal{T}'}\Phi(x;w)$. Then we have

$$\|e - d\Phi(x;w)\|_F^2 \leq \|e\|_F^2 \ ,$$

i.e.,

$$\|d\Phi(x;w)\|_F^2 \leq 2\langle e, d\Phi(x;w)\rangle \ .$$

Now observe that

$$2\langle e, d\Phi(x;w)\rangle \leq 2\|d\|_F\|\Phi(x;w)e^T\|_F$$

and

$$\begin{aligned}
\|d\Phi(x;w)\|_F^2 &= \operatorname{trace}(d\Phi(x;w)\Phi(x;w)^T d^T) \\
&= \sum_{t=1}^{T} d_{t,\cdot}^T \Phi(x;w)\Phi(x;w)^T d_{t,\cdot} \\
&\geq \lambda_{\min} T \sum_{t=1}^{T} \|d_{t,\cdot}\|_2^2 \\
&= \lambda_{\min} T \|d\|_F^2 \ .
\end{aligned}$$

Therefore,

$$\lambda_{\min} T \|d\|_F^2 \leq 2\|d\|_F\|e\Phi(x;w)^T\|_F \ ,$$

or, rearranging terms and substituting for d and e,

$$\begin{aligned}
\|\Pi_{\mathcal{T}'} - \Pi_{\mathcal{T}}\|_F &\leq \frac{2}{\lambda_{\min} T}\|(\Phi(x;w) - \Pi_{\mathcal{T}'}\Phi(x;w))\Phi(x;w)^T\|_F \\
&\leq \frac{2}{\lambda_{\min} T}\|\Phi(x;w)\|_F\|\Phi(x;w) - \Pi_{\mathcal{T}'}\Phi(x;w)\|_F \ .
\end{aligned}$$

□

From this result we have the following corollary:

Corollary 74. *Define $\hat{w}, w^\star, \hat{\mathcal{T}}$, and $\mathcal{T}^\star$ as in Section 4.5.3. Assume that the smallest eigenvalue of the normalized Gram matrix $\Phi(x;\hat{w})\Phi(x;\hat{w})^T/T$, $\lambda_{\min}$, satisfies $\lambda_{\min} > 0$. Then*

$$d_F(\hat{\mathcal{T}}, \mathcal{T}^\star) \leq \frac{2}{\lambda_{\min} T} \|\Phi(x;\hat{w})\|_F \|\Phi(x;\hat{w}) - \Pi_{\mathcal{T}^\star}\Phi(x;\hat{w})\|_F \ .$$

Proof. Note that $\mathcal{L}(\hat{\mathcal{T}}, \hat{w}; x) \leq \mathcal{L}(\mathcal{T}^\star, \hat{w}; x)$. The result then follows directly from Proposition 73. □

Next, we provide a lower bound on the distance between segmentations.

Proposition 75. *Let $\mathcal{T}$ and $\mathcal{T}'$ be two segmentations of a sequence x with feature map ϕ parameterized by w. A lower bound on the Frobenius norm distance between the segmentations is given by*

$$d_F(\mathcal{T}, \mathcal{T}') \geq \frac{1}{\|\Phi(x;w)\|_F} \left| \|\Phi(x;w) - \Pi_{\mathcal{T}}\Phi(x;w)\|_F - \|\Phi(x;w) - \Pi_{\mathcal{T}'}\Phi(x;w)\|_F \right| \ .$$

Proof. Observe that

$$\begin{aligned}
&\left| \|\Phi(x;w) - \Pi_{\mathcal{T}}\Phi(x;w)\|_F - \|\Phi(x;w) - \Pi_{\mathcal{T}'}\Phi(x;w)\|_F \right| \\
&\leq \|(\Phi(x;w) - \Pi_{\mathcal{T}}\Phi(x;w)) - (\Phi(x;w) - \Pi_{\mathcal{T}'}\Phi(x;w))\|_F \\
&= \|\Pi_{\mathcal{T}'}\Phi(x;w) - \Pi_{\mathcal{T}}\Phi(x;w)\|_F \\
&\leq \|\Phi(x;w)\|_F \|\Pi_{\mathcal{T}'} - \Pi_{\mathcal{T}}\|_F \\
&= \|\Phi(x;w)\|_F d_F(\mathcal{T}, \mathcal{T}') \ .
\end{aligned}$$

□

From this we have the following corollary.

Corollary 76. *Define $\hat{w}, w^\star, \hat{\mathcal{T}}$, and $\mathcal{T}^\star$ as above. A lower bound on the Frobenius norm distance between the segmentations $\hat{\mathcal{T}}$ and $\mathcal{T}^\star$ is given by*

$$d_F(\hat{\mathcal{T}}, \mathcal{T}^\star) \geq \frac{1}{\|\Phi(x;\hat{w})\|_F} \left| \|\Phi(x;\hat{w}) - \Pi_{\hat{\mathcal{T}}}\Phi(x;\hat{w})\|_F - \|\Phi(x;\hat{w}) - \Pi_{\mathcal{T}^\star}\Phi(x;\hat{w})\|_F \right| .$$

C.4 Datasets

In this appendix we describe the datasets used in this chapter, including what their features are, how they were generated, and how they are preprocessed. In every case we define a change point to be the *first* observation in each segment.

C.4.1 Synthetic data

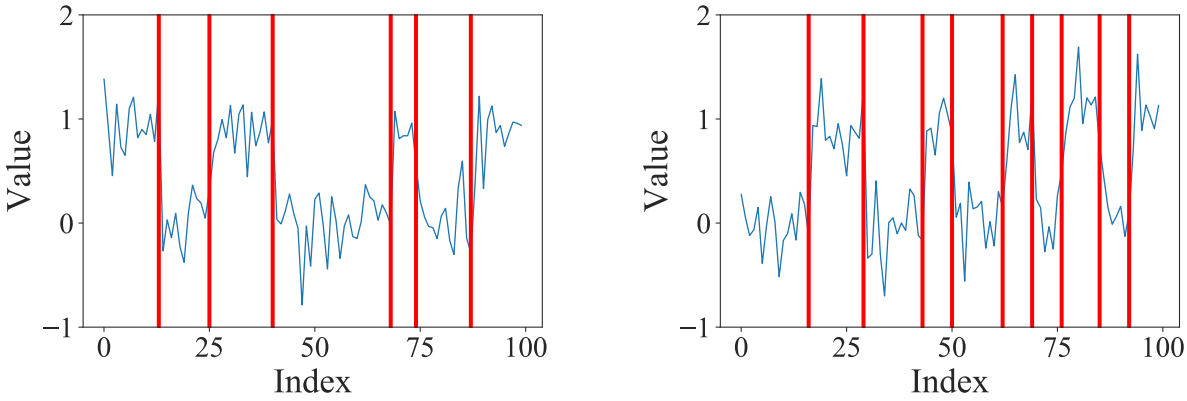
The synthetic datasets we generate consist of 1D sequences with observations in each segment drawn from either $N(0, 2^{-4})$ or $N(1, 2^{-4})$ and then transformed using the function

$$f(x) = \sum_{\ell=1}^{3} w_{\ell,3} \exp\left(-\frac{1}{2}(w_{\ell,1}x + w_{\ell,2})^2\right) ,$$

where each of the w's was drawn independently from $N(0, 9)$. The label of each observation is the mean of the distribution from which it was drawn.

To generate a sequence we first draw a number of change points m between 1 and 10. We then sequentially select the locations for the change points uniformly at random, subject to the constraint that each new change point must lie at least 5 indices away from every previously selected change point, in addition to the two end points. We choose the first segment label in each sequence uniformly at random and thereafter the labels alternate between segments. Finally, we draw observations with the given labels from the aforementioned distributions and transform them by passing them through the function f. In total there are 500 training sequences, 100 validation sequences, and 100 test sequences for each random seed.

Two example sequences are displayed in Figure C.1. There are a couple of points to note about the dataset. First, while the number of change points was chosen uniformly at random,



(a) Training set sequence 1

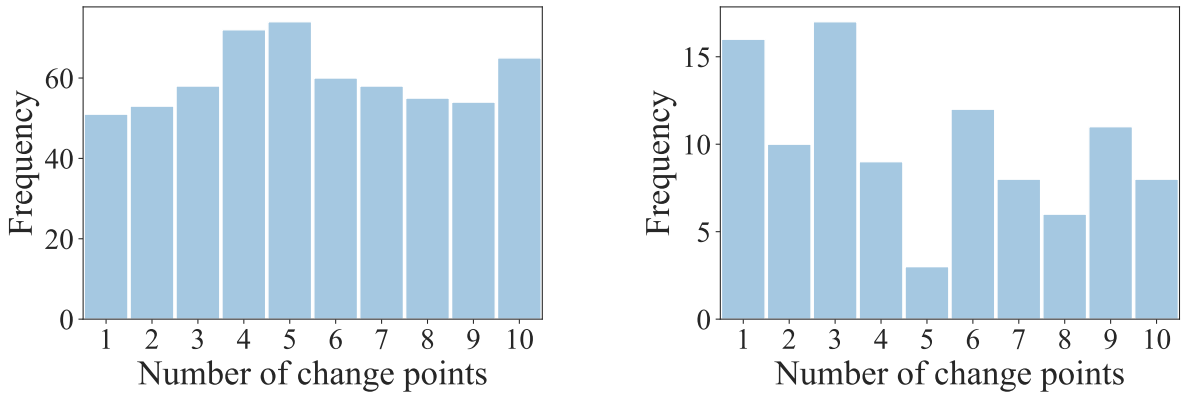
(b) Training set sequence 2

Figure C.1: Two example sequences from the synthetic dataset. The goal is to estimate the indices (indicated by the vertical red lines) at which changes in the mean occur in the untransformed data.

we can see from Figure C.2 that there is still variation in the frequency with which each was chosen. This is especially evident in the test set, in which there are only 100 total sequences. Second, from Figure C.3 we can see that the change-point locations are relatively uniform between 5 (the minimum allowed change point index) and 95 (the maximum allowed change point index). Note that each of these plots was created using the seed 0 for randomization when generating the dataset. Some variation in the plots exists when changing the random seed.

C.4.2 MNIST-seq

The MNIST dataset (LeCun et al., 2001) consists of black-and-white images of handwritten digits, along with their labels. Each image, 28×28 pixels in size, displays one digit between zero and nine. The original dataset contains 60,000 training images and 10,000 test images. To build an analogous dataset for change-point estimation we generate sequences of length



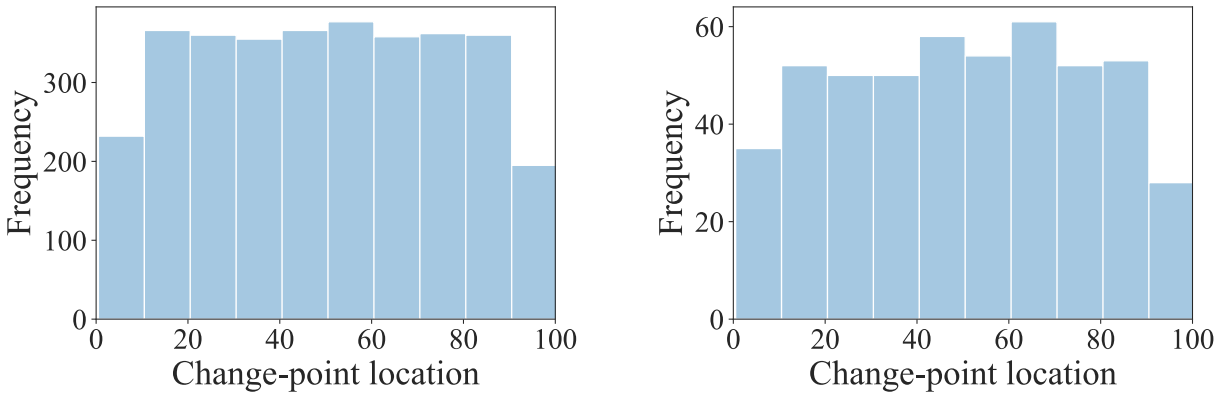
(a) Combined training and validation sets

(b) Test set

Figure C.2: Distribution of the number of change points in the synthetic dataset.

100 consisting of MNIST images, creating a dataset we call MNIST-seq. Within each sequence there are runs of images of the same digit. When performing change point estimation on the dataset the goal is to be able to estimate the locations of changes in the labels of the digits in each sequence.

The sequences are generated in the same manner as for the synthetic data, with the following additions. First, given the change points, we draw a label for each segment uniformly at random, subject to the constraint that sequential segments must have different labels. Second, we draw images with the given class labels uniformly at random but without replacement to generate each sequence. Due to edge cases, as no image is ever reused, the 60,000 training images and 10,000 test images do not always generate 600 training and 100 test sequences. Moreover, we hold out 100 sequences as a validation set. Therefore, for the random seeds we use in the experiments the training set contains between 486 and 500 sequences, the validation set contains 100 sequences, and the test set contains between 98 and 100 sequences. We preprocess each sequence by subtracting the mean and dividing by the standard deviation computed across all pixels in the combined training and validation sets.



(a) Combined training and validation sets

(b) Test set

Figure C.3: Distribution of the locations of the change points in the synthetic dataset.

Figure C.4 displays two subsequences of an example sequence from MNIST-seq. The change points are located at indices 6 and 5, respectively, when using 1-based indexing. Figure C.5 displays the mean and standard deviation of each pixel (after having applied the aforementioned global standardization) in the combined training and validation sets. The corresponding plots for the test set look similar. It is noteworthy that not all pixels vary the same amount across the dataset. The distributions of the number and locations of the change points look similar to those of the synthetic data.

C.4.3 Honey Bee Waggle Dance

Certain species of honey bees engage in a "waggle dance" when they are within a certain distance of a food source (von Frisch, 1993). This dance, used to indicate to other bees the location of the target, consists of three parts: (1) the waggle, in which the tail of the bee oscillates from side to side as the bee moves forward; (2) turn left; and (3) turn right. The dance occurs in what is roughly a figure-eight shape, as the bee alternates between waggling, turning right until it roughly reaches the location where it started waggling, waggling again,

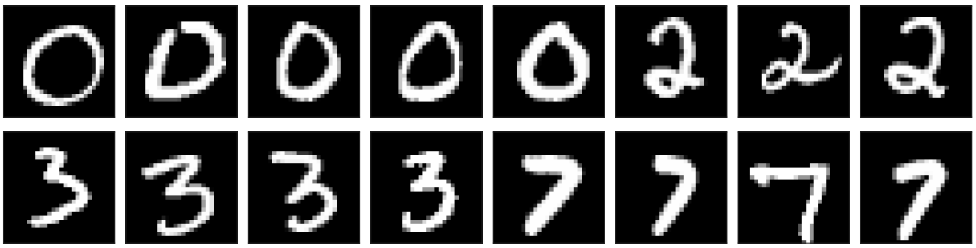


Figure C.4: Two example subsequences from an MNIST-seq sequence of images. The goal is to estimate the indices at which changes in the digit labels occur.

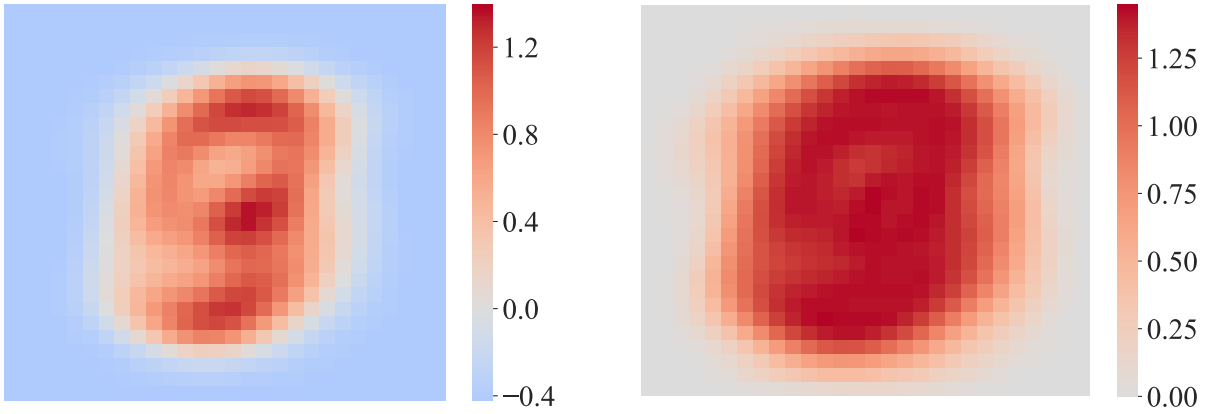
and then turning left until it roughly reaches the location where it started waggling. The direction of the waggle indicates the direction of the target relative to the location of the sun. Moreover, the length of the waggle is positively correlated with the distance to the target.

The Honey Bee Waggle Dance dataset of Oh et al. (2008) consists of six sequences of measurements derived from videos of dancing honey bees. Each sequence, from a single honey bee, consists of the (x, y) locations of the bee, along with the angle θ of the bee's head at each frame of the video. The goal of change-point estimation on this dataset is to determine when there are changes between the turning and waggling phases.

We use the version of the dataset from Chang et al. (2019).[1] In this version of the dataset the angle θ was transformed into the difference between $\cos(\theta)$ and $\sin(\theta)$. Moreover, each feature was scaled to lie in $[0, 1]$. To be consistent with Chang et al. (2019), we make the first four data files the training set, the fifth the validation set, and the sixth the test set.

The features from each file in the dataset, along with the change points (in red), are shown in Figure C.6. From top to bottom, the displayed features in each plot correspond to the x-coordinate, y-coordinate, and angle difference. There are several important qualities to note. First, the length of each time series differs and varies between 602 and 1124. Next, the trends within each time series differ, and the frequencies of the oscillations differ. This means

[1] `https://github.com/OctoberChang/klcpd_code/tree/master/data/beedance`



(a) Mean (b) Standard deviation

Figure C.5: Statistics of each pixel across the combined training and validation sets of MNIST-seq.

that there are many more change points for some time series (28 for the validation set) than for others (15 for the test set). Prior to running change-point estimation we difference the data and generate sliding windows as described in Section 4.6.

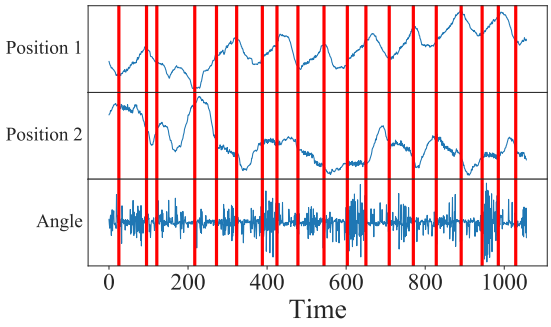
C.5 Additional Training Details

In this section we provide details concerning the parameter settings for our algorithm, XSCPE, and the alternative algorithm we compare to, KCPE.

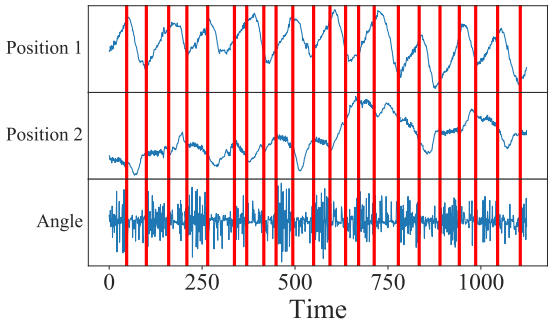
C.5.1 Parameter settings for XSCPE

Between the model, the training objective, and the optimization method, there are many parameters that need to be set for our XSCPE algorithm.

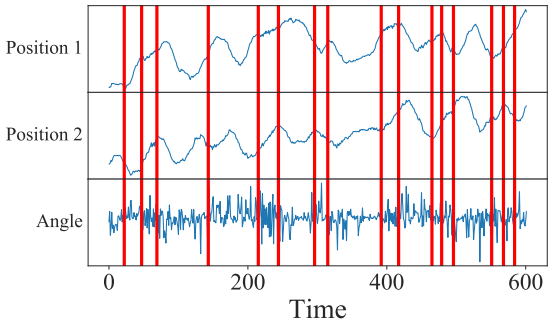
Fixed parameters. We begin by describing the parameters that are fixed throughout the experiments. The network used for the synthetic data is the same as the true network, but the parameters are initialized by drawing independent values from $N(0, 1)$. For MNIST-seq



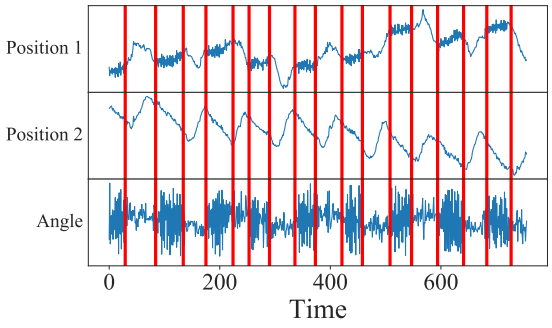
(a) Training set time series 1



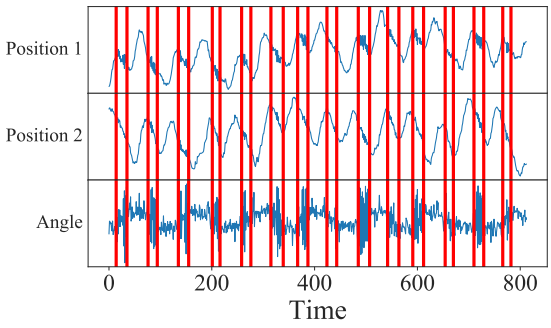
(b) Training set time series 2



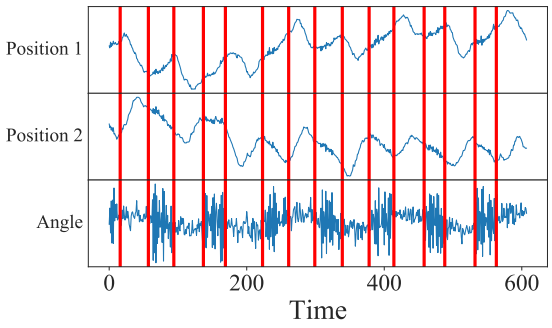
(c) Training set time series 3



(d) Training set time series 4



(e) Validation set time series



(f) Test set time series

Figure C.6: Features from the six bee dance files.

and the Bee Dance dataset we use the LeNet-5 convolutional kernel network (CKN) and a fully-connected kernel network, respectively, each with 32 filters per layer (see Chapter 2). Their filters are initialized bottom-up in the network by choosing 32 feature maps uniformly at random from the output of the previous layer. Each kernel is approximated using the Nyström method with a regularization of 0.001, as done by Mairal (2016). The matrix inverse square root term in the Nyström method is computed using 20 iterations of the intertwined Newton method discussed in Chapter 2. The features output by the CKNs are not transformed via e.g., normalization or standardization. During the training the batch size is set to the entire dataset for the synthetic and Bee Dance datasets. For MNIST-seq the batch size is set to 50 sequences due to GPU memory limitations. The training lasts for 100 iterations. When applying Algorithm 6 we set the minimum distance between change points to be $\Delta = 1$.

Hold-out validation. For the kernel network on the Bee Waggle Dance dataset we first simultaneously tune the amount of differencing over the set {no differencing, first differences, second differences}, the sliding window size over the values $\{1, 3, 5, 7\}$, the depth of the network over the values $1, 2, \ldots, 10$, and the bandwidth over the values $0.1, 0.2, \ldots, 1.0$ on untrained networks. The best performance on the validation set in terms of the Frobenius distance is attained with first differences, a sliding window of size three, a depth of five, and a bandwidth of 0.5. For the LeNet-5 CKN on MNIST-seq we first tune the bandwidth over the values $0.1, 0.2, \ldots, 1.0$ on untrained networks. The best performance on the validation set in terms of the Frobenius distance is attained with a bandwidth of 0.7. For the RBF network on the simulated data, the network is fixed and the bandwidth parameters are learned.

As the number of remaining hyperparameters is large, we tune them sequentially as follows. The most essential hyperparameter to tune is the penalty λ on the square root of the sum of the segment variances. Therefore, we first tune this parameter over the values 2^i for $i = -10, -9, \ldots, 10$ when fixing the learning rate to be 2^{-4} (for the RBF network and the LeNet-5 CKN) or 2^{-8} (for the fully-connected kernel network), γ to be either 2^{-4} (for the RBF network) or 0 (for the fully-connected kernel network and the LeNet-5 CKN), and ϵ to

be 10^{-4}. Following this, we tune the learning rate over the values 2^i for $i = -10, -9, \ldots, 10$ for fixed γ and ϵ as above. Then we tune γ for the RBF network over the values 2^i for $i = -10, -9, \ldots, 10$ for fixed ϵ. The final parameter we tune when performing alternating optimization is ϵ, which is tuned over the values 10^i for $i = -5, -4, \ldots, 0$. When performing optimization via smoothing for the synthetic data experiments we set α to be the value from the set 10^i for $i = -5, -4, \ldots, 5$ leading to the largest difference between the non-smoothed and smoothed training objective value vs. iteration curves. For the other experiments, since the smoothing never performed better, we set it to be the smallest value leading to a noticeable difference between the non-smoothed and smoothed curves.

C.5.2 Parameter settings for KCPE

We use the KCPE method of Harchaoui and Cappé (2007) with the Gaussian RBF kernel. As such, there is only one parameter to be set: the kernel bandwidth. We set the bandwidth using hold-out validation over the values $\{0.1, 0.2, 0.3, \ldots, 2.0\}$ for the simulated and Bee Waggle Dance datasets. For MNIST-seq we use hold-out validation over the values $\{10, 20, 30, \ldots, 200\}$.

C.6 Additional Experimental Results

In this appendix we present additional experimental results expanding on those from Section 4.6. First, in Figure C.7 we perform a sensitivity analysis of the LeNet-5 CKN on MNIST-seq. From the figure we can see that the algorithm is most sensitive to the parameter λ. There is only a small window, from 4 to 16, in which the resultant average Frobenius distance is within a factor of two of the optimal average Frobenius distance.

Next, we further examine the effect of smoothing the objective. Figure C.8 shows the effect of decaying α across iterations, with $\alpha_a = \alpha/a$ at iteration a. We once again find for the synthetic experiment that $K = 10$ seems to outperform $K = 1$, but only after approximately 30 iterations. Moreover, there is little to no difference between smoothing and not smoothing for the MNIST-seq and Bee Waggle experiments. Figures C.9 and C.10 show how the average

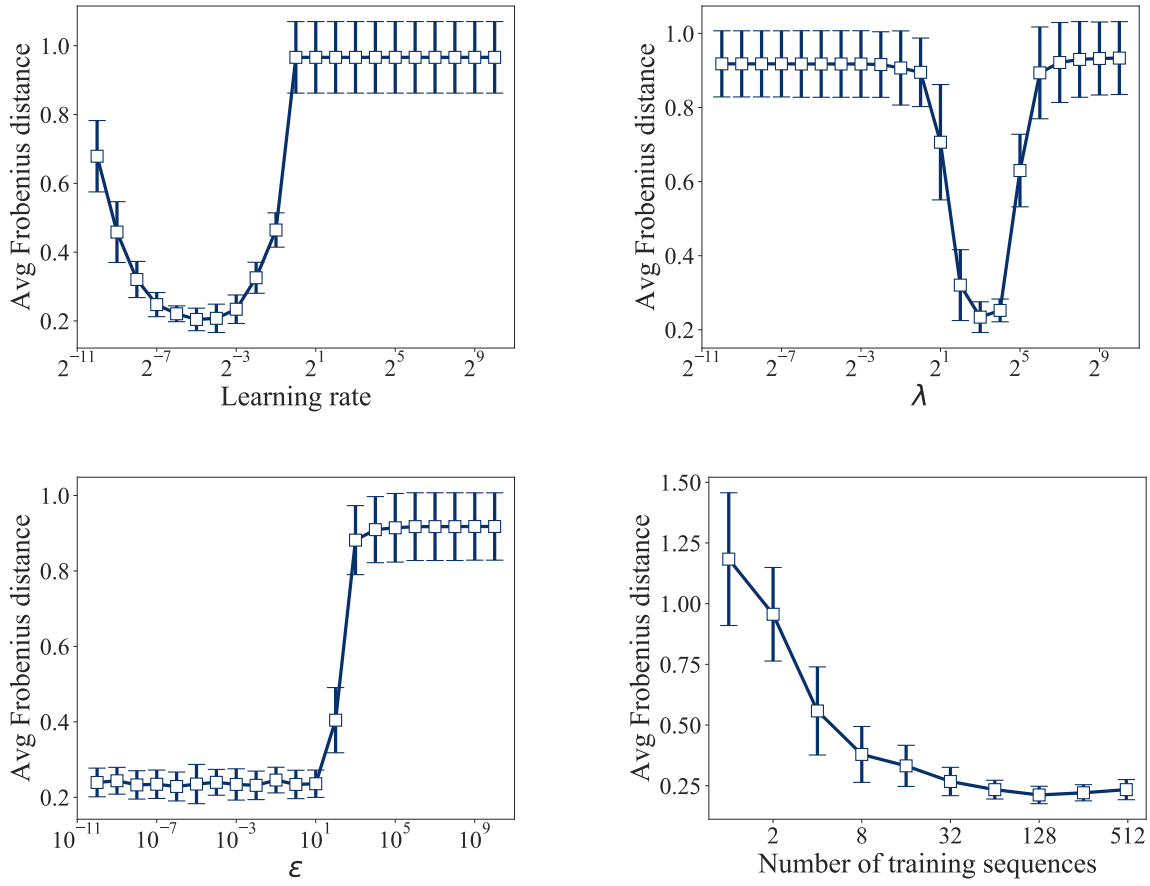


Figure C.7: Sensitivity analysis of the parameters when training a LeNet-5 CKN on MNIST-seq with no labeled training sequences. The error bars represent one standard deviation across 10 trials with different random seeds.

number of non-zero entries in the vector $\text{proj}_{\Delta^{K-1}}(-\mathcal{L}(w; x^{(i)})_{[1:K]}/\alpha)$ varies across iterations. We can see that the number of non-zero entries often decreases as the objective value decreases. Moreover, the decrease is larger when α is decayed across iterations, which we would expect based on Proposition 3 of Pillutla et al. (2019).

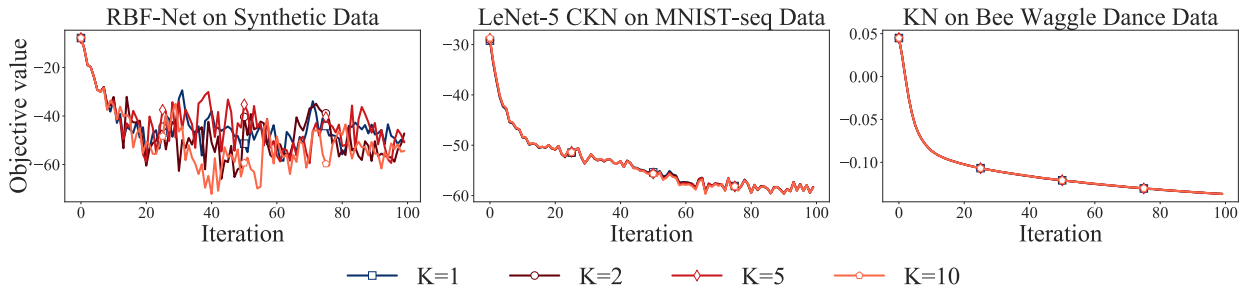


Figure C.8: Average objective value on the training sets when varying the value of K in the top-K heuristic. The objective values are averaged across 10 trials with different random seeds. The smoothing parameter α was initialized to 10, 1000, and 1, respectively, for the simulated, MNIST-seq, and Bee Dance experiments. It was then decayed across iterations and given by $\alpha_a = \alpha/a$ at iteration a.

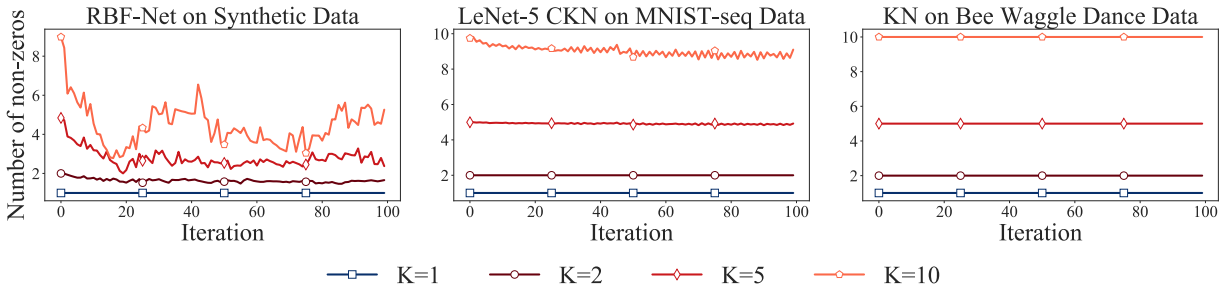


Figure C.9: Average number of non-zero entries in $\text{proj}_{\Delta^{K-1}}(-\mathcal{L}(w; x^{(i)})_{[1:K]}/\alpha)$ when varying the value of K in the top-K heuristic. The number of non-zero values is averaged across all training sequences and across 10 trials with different random seeds. The smoothing parameter α was fixed across iterations to 10, 1000, and 1, respectively, for the simulated, MNIST-seq, and Bee Dance experiments.

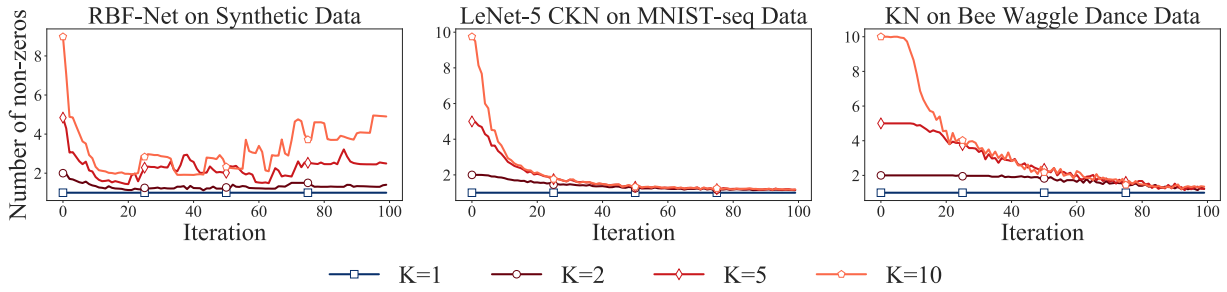


Figure C.10: Average number of non-zero entries in $\text{proj}_{\Delta^{K-1}}(-\mathcal{L}(w; x^{(i)})_{[1:K]}/\alpha)$ when varying the value of K in the top-K heuristic. The number of non-zero values is averaged across all training sequences and across 10 trials with different random seeds. The smoothing parameter α was initialized to 10, 1000, and 1, respectively, for the simulated, MNIST-seq, and Bee Dance experiments. It was then decayed across iterations and given by $\alpha_a = \alpha/a$ at iteration a.

Appendix D

APPENDIX FOR CHAPTER 5

D.1 Additional Experimental Details

In this appendix we provide additional details regarding the data used and the annotation of the physical data.

D.1.1 Data

We use the physical data from 18 cruises and the biological data from 9 cruises. The cruises and some of their characteristics are listed in Table D.1. The 18 cruises are chosen to be all of the cruises that are neither coastal cruises nor cruises around the Hawaiian islands and for which physical data exists. The subset of nine cruises consists of those from the 18 for which biological data is available.

The data is cleaned as follows. For the physical data we first remove the dates that are not in chronological order. Next, we remove observations for which the temperature is unavailable or larger than 60°C. Furthermore, we remove observations for which the salinity is less than 10 or larger than 60 PSU. Finally, we remove successive files that were recorded at times when the ship had not moved more than 0.1 km. For the biological data we first remove files for which the physical data is not available. We also delete the entries corresponding to beads rather than phytoplankton.

D.1.2 Manual annotation of physical data

Changes in phytoplankton community structure often occur when different water parcels mix together. A mixing event of different water masses can be identified by studying changes in water temperature and salinity. The mass of a water parcel (defined by its temperature

Cruise	Location	Length (km)	# changes	Physical	Biological
KM1314	Seattle - Hawaii	6081	38	✓	
KM1502	Portland - Hawaii	3987	44	✓	✓
KM1712	Hawaii - Alaska	5910	48	✓	✓
KM1713	Alaska - Hawaii	4458	27	✓	✓
KM1906-tf	Subtropical front	1233	27	✓	
KN210-04	South Atlantic	17436	92	✓	✓
KOK1606	Subtropical front	4006	60	✓	✓
MGL1704	Subtropical front	4939	32	✓	✓
RR1814	Seattle - Hilo	6599	30	✓	
RR1815	Hilo - Papeete	5540	49	✓	
TN248	Gulf of Alaska	2049	28	✓	✓
TN257	Seattle - Hawaii	4499	35	✓	
TN271	Seattle - Hawaii	4527	41	✓	✓
TN292	Seattle - Hawaii	3905	30	✓	✓
Tokyo_1	North Pacific	11384	83	✓	
Tokyo_2	North Pacific	11485	48	✓	
Tokyo_3	North Pacific	11664	118	✓	
Tokyo_4	North Pacific	5885	81	✓	

Table D.1: Details of the cruises used. The column "# changes" provides the number of change points in the annotated physical data. The last two columns indicate whether the physical data and biological data, respectively, from the cruise are used.

and salinity) can change when mixing with other water masses and when in contact with the surface, where heat and fresh water can be gained or lost. Here, we assume that large changes in water temperature and salinity measured around 5-m depth during the cruise were caused by mixing of different water masses while small changes were caused by air/sea interactions. For each cruise, water mixing events were manually annotated halfway along lines that connect two clusters of points in a temperature-salinity diagram.

D.2 Additional Experimental Results

In this appendix we present additional experimental results. First, Figure 5.3 overlays the ten estimated change points from the KOK1606 cruise on plots of the latitude and longitude of the ship. Three of the change points located in the data from the trip north also appear to

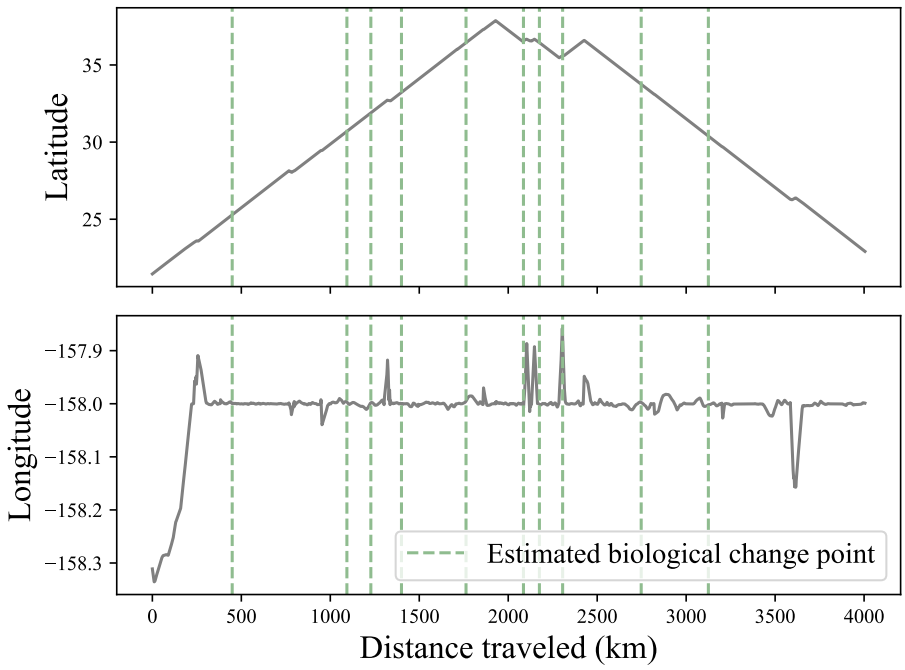
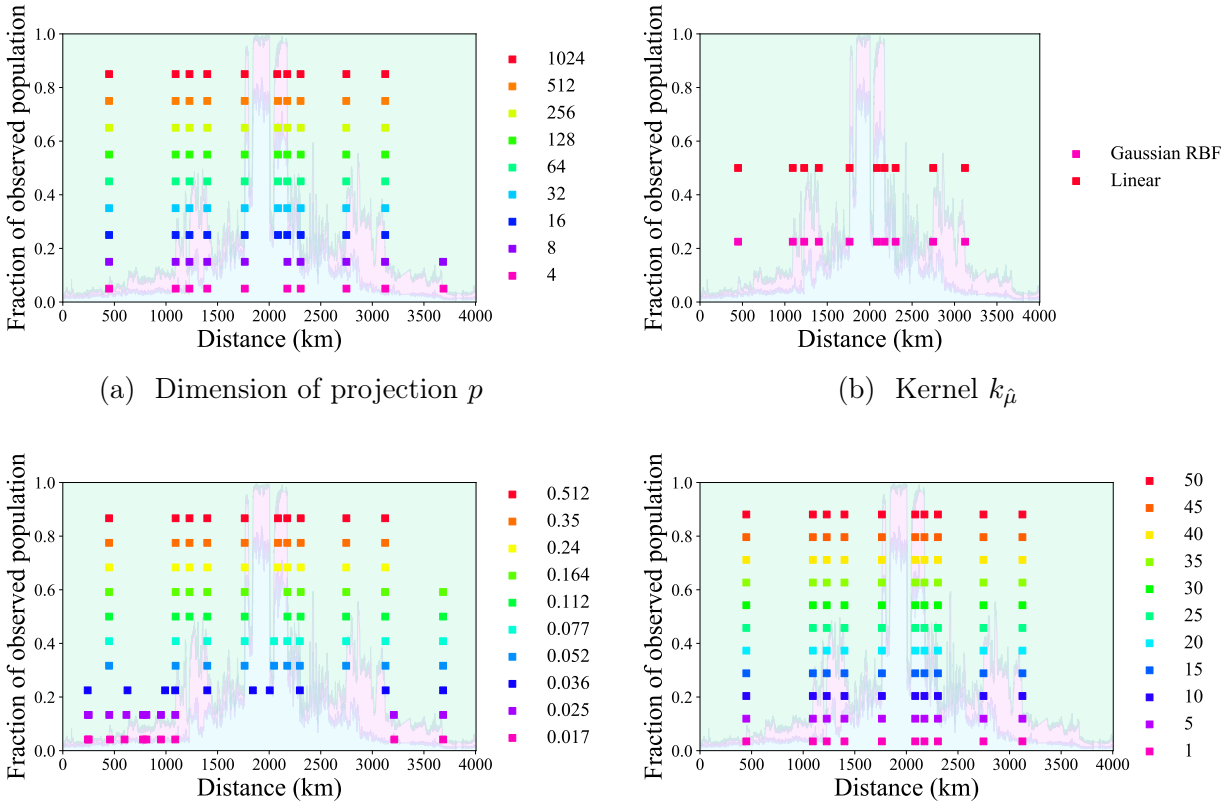


Figure D.1: Estimated change points on the biological data from the KOK1606 cruise overlaid on the latitude and longitude of the ship throughout the cruise.

exist in the data from the trip south: those from the trip north are all within 60km in space of those on the trip south. As noted in Section 5.4, we expect that the changes points could have shifted in space by 60km over the course of the cruise due to the ocean currents.

Next, Figure D.2 assesses how sensitive our method is to the choice of hyperparameters. We can see that in general it is not sensitive to the dimension of the projection, the choice of kernel $k_{\hat{\mu}}$, or the minimum allowable distance between change points. The bandwidth $\sigma_{\hat{\mu}}$ only results in poorly-estimated change points if it is too small (e.g., less than 0.05).

Figure D.3 provides additional results related to estimating the number of change points. In the left panel we plot the estimated vs. annotated number of change points in the physical data when using the rule of thumb from Harchaoui and Lévy-Leduc (2007). As we can see, there is no correlation between the estimated and annotated number of change points. In the right panel we plot a histogram of normalized distances from the annotated physical change



(a) Dimension of projection p

(b) Kernel $k_{\hat{\mu}}$

(c) Kernel bandwidth $\sigma_{\hat{\mu}}$

(d) Minimum distance between change points

Figure D.2: Sensitivity of the estimated change points on the biological data from the KOK1606 cruise to changes in the parameters of the method.

points to the nearest estimated biological change points. The normalized distance from the annotated change points to the estimated change points is less than 5% for 18% of the change points and is less than 10% for 29% of the change points.

Finally, Figure D.4 displays the estimated biological change points and the annotated physical change points for the KOK1606 cruise. For this cruise the estimation of the number of change points works well: the estimated number of biological change points is 64, whereas the number of annotated physical change points is 60.

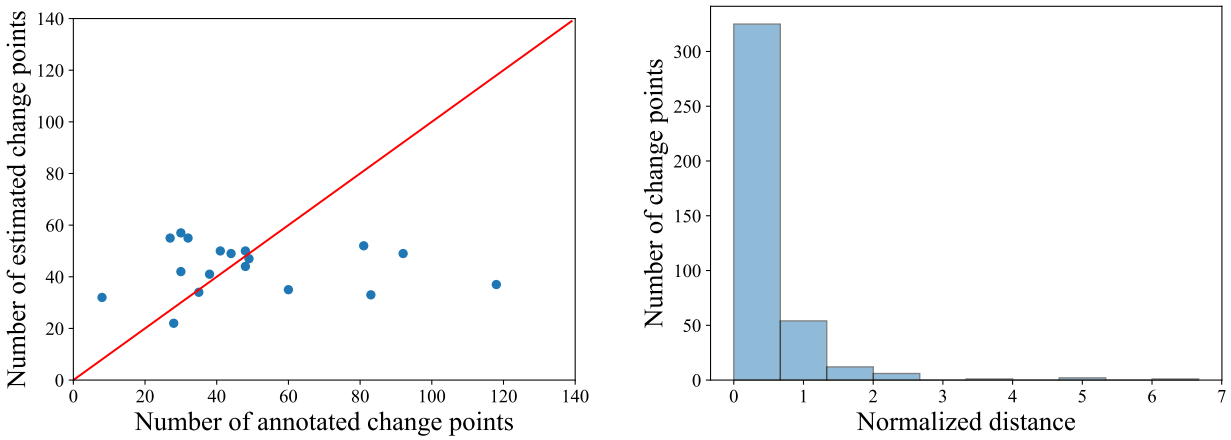


Figure D.3: Estimated and annotated number of change points on the physical data based on the rule of thumb from Harchaoui and Lévy-Leduc (2007) (left) and histogram of the minimum distances from each annotated physical change point to the nearest estimated biological change point for the same cruise, normalized by the average distance between estimated change points within the cruise (right). The diagonal red line in the plot on the left denotes the locations where the points would ideally lie.

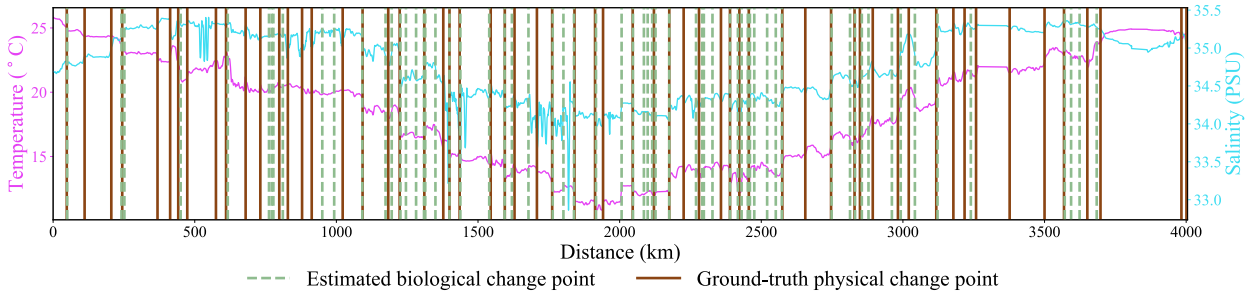


Figure D.4: Estimated change points in the biological data and annotated change points in the physical data from the KOK1606 cruise overlaid on the temperature and salinity data recorded throughout the cruise.