Robust Submodular Partitioning and Linear Models of Deep ReLU Networks

Shengjie Wang

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

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Jeff Bilmes, Chair

Ali Farhadi

Kevin Jamieson

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Robust Submodular Partitioning and
Linear Models of Deep ReLU Networks

Shengjie Wang

Chair of the Supervisory Committee:
Professor Jeff Bilmes
Department of Electrical Engineering

Machine learning models, especially deep neural networks, have achieved great success in numerous real-world tasks. As we achieve better performance with larger models, one major challenge emerges that the costs of training machine learning systems become expensive and even prohibitive. Also, the deep learning model works as a block box in many applications with little interpretation of its behaviors. In this dissertation, we investigate two problems: 1) partitioning of training data into diverse and representative blocks for gradient computation to get improved efficiency and performance for machine learning models and 2) decomposition of ReLU deep neural networks as a collection of linear models for data points and we utilize the linear models to better understand and improve the network performance.

For the **first part** of the thesis, we first investigate the problem of partitioning the training dataset into multiple blocks which are equally diverse. The theoretical abstraction of the problem is denoted as robust submodular partitioning. In robust submodular partitioning, we aim to allocate a set of items into m blocks, so that the evaluation of the minimum block according to a submodular function is maximized. Robust submodular partitioning promotes the diversity of every block in the partition. It has many applications in training machine learning models, e.g., partitioning data into blocks for distributed training so that the gradients computed for every block are consistent. We study the robust submodular par-

titioning problem and give an efficient Min-Block Greedy algorithm with a $1/m$ guarantee. We further study an extension of the robust submodular partition problem with an additional constraint (e.g., cardinality, multiple matroids, or knapsack) on every block. For example, when partitioning data for distributed training, we can add a constraint that the number of samples of each class is the same in each partition block, making the partitioned data balanced. We present two classes of algorithms, i.e., Min-Block Greedy based algorithms ($\Omega(1/m)$ bound), and Round-Robin Greedy based algorithms (constant bound) and show that under various constraints, they still have good approximation bounds. We further investigate the robust submodular partitioning problem under cardinality constraint and apply it to generate high-quality mini-batches for stochastic gradient methods. With computational hardware (e.g., GPUs) getting dramatically faster over time, sampling a mini-batch of data points uniformly at random becomes less practical, as randomly accessing data points from disk can be slow, leading to a bottleneck for modern machine learning systems. In practice, datasets are typically written to disk according to an arbitrarily generated sequence of indices. This makes sequential access of this chosen order possible with low overhead compared to random access. On the other hand, there is a chance that the sequence is poor for training, and since it is fixed over multiple iterations of training, performance can suffer. We prove better bounds of the Min-Block Greedy algorithm for this case and greatly reduce the memory/computation costs by applying hierarchical partitioning. We compare our deterministically generated mini-batch sequences to randomly generated sequences and show that the deterministic sequences significantly beat the mean and worst performance of random sequences, and often outperform the best of the random sequences.

For the **second part** of the thesis, we focus on understanding and improving the ReLU deep network through its decomposition as a linear model for every data point. A ReLU deep network (or more generally for deep networks with piecewise linear activation functions) is essentially a piecewise linear model. Therefore, the model is locally linear around

every data point, and the linear model weights are equal to the gradient of the network output with respect to its input data point. Based on this observation, we first introduce the Extended Data Jacobian Matrix (EDJM) as an architecture-independent tool to analyze neural networks at the manifold of interest. For ReLU networks, the EDJM is essentially a collection of linear models for all data points, represented as a matrix. The spectrum of the EDJM is found to be highly correlated with the complexity of the learned functions. After studying the effect of dropout, ensembles, and model distillation using EDJM, we propose a novel spectral regularization method that improves network performance. However, we note that such a regularization method has greatly increased computational costs, limiting its practical usage. Next, we show an efficient regularization method Jumpout, an improved version of dropout, based on linear models of ReLU networks. We discuss three novel insights about dropout for DNNs with ReLUs: 1) dropout encourages each local linear piece of a DNN to be trained on data points from nearby regions; 2) the same dropout rate results in different (effective) deactivation rates for layers with different portions of ReLU deactivated neurons; and 3) the rescaling factor of dropout causes a normalization inconsistency between training and test when used together with batch normalization. The above leads to three simple but nontrivial modifications resulting in our method "Jumpout." Jumpout significantly improves the performance of different neural nets on multiple datasets, while introducing negligible additional memory and computation costs. Finally, we aim to explain the network behavior based on the linear model for every data point, particularly based on the bias term of the linear model. The gradient of a deep neural network (DNN) w.r.t. the input provides information that can be used to explain the output prediction in terms of the input features and has been widely studied to assist in interpreting DNNs. In a linear model (i.e., $g(x) = wx + b$), the gradient corresponds to the weights w. The bias b, however, is usually overlooked in attribution methods. We observe that since the bias in a DNN also has a non-negligible contribution to the correctness of predictions, it can also play a significant

role in understanding DNN behavior. We propose a backpropagation-type algorithm "bias back-propagation (BBp)" that starts at the output layer and iteratively attributes the bias of each layer to its input nodes as well as combining the resulting bias term of the previous layer. Together with the backpropagation of the gradient generating w, we can fully recover the locally linear model $g(x) = wx + b$. In experiments, we show that BBp can generate complementary and highly interpretable explanations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Firstly, I would like to thank my Ph.D. adviser Professor Jeff Bilmes. Jeff has been a great mentor to me in research. He taught me about machine learning and submodularity from the classic results to the cutting-edge research. His mathematical intuition and practical ideas always bring new insights to my research. Jeff is also very strict on details, where I benefit a lot. I could not count how many times he has identified essential bugs or mistakes from details in my codes, plots and proofs. Under his influence, I have developed much better habits about recording details and thinking more rigorously, which I believe would be critical for my future career as a machine learning researcher. Besides research, Jeff has also been a very supportive friend. I enjoy discussing with Jeff about many random topics such as music, cooking, culture, and etc. Overall, I feel extremely lucky to have Jeff as my Ph.D. advisor. Thank you, Jeff!

I would also like to thank my thesis committee members: Prof. Ali Farhadi, Prof. Kevin Jamieson, and Prof. Archis Ghate. I appreciate their constructive questions and insightful suggestions on my research. Their help is invaluable to this thesis.

I would also like to thank my collaborators – Tianyi Zhou, Wenruo Bai, Chandrashekhar Lavania, Lily Kumari, Wei Kai, Rishabh Iyer, John Halloran, Sunil Thulasidasan, Arnav Das and Gantavya Bhatt. It has been an amazing experience working with all of them. Without their help, it is impossible to accomplish the research I present in the thesis. Besides research, they have been helpful and sincere friends to me, adding joyful colors to my Ph.D. experience. I would also like to thank my mentors during my internship at Microsoft Research: Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matthew Richardson. Without their guidance, I would not have a chance to think more deeply about neural networks, which

constitutes an important part of the thesis.

Finally, I would like to dedicate this thesis to my family: my love Xinan Wang and my parents Lin Wang and Xiaoyan Zhu. It is hard to imagine my Ph.D. journey without their support and accompany.

DEDICATION

to Xinan Wang and my parents Lin Wang and Xiaoyan Zhu for their unending support.

Part I

ROBUST SUBMODULAR PARTITIONING

In the first part of this thesis, we focus on the problem of robust submodular partitioning of the training dataset, where we partition the data into multiple blocks, and every block is equally diverse and representative. We study theoretical perspectives of the problem in both unconstrained and constrained cases and propose many algorithms with theoretical guarantees. From the practical point of view, the partitioning can be used to generate blocks for distributed training or mini-batches for stochastic gradient methods. Empirically, we test the mini-batches generated by our robust submodular partitioning methods and get improved performance compared to randomly generated mini-batches.

Chapter 1

INTRODUCTION

The problem of partitioning a given set V of items into m blocks, where any two blocks share no items in common, arises in many real-world scenarios and machine learning applications. For example, clustering is a form of data partitioning to generate blocks of data such that samples within every block are similar and redundant. Moreover, for the task of distributed training on multiple machines, we aim to partition the training dataset into blocks such that the data distribution across different blocks is consistent, or in other words, every partitioned block is representative of the entire training dataset.

As an optimization problem, partitioning aims to generate the blocks so that the utilities of the blocks, as measured by a given set function, are good. By maximizing a submodular utility function for each partitioned block, we encourage each block to be representative of the ground set V. We use submodular optimization to describe and enforce the representativeness of the selected set of training data samples. Submodular functions, often used in economics, operations research, or (more recently) machine learning, are a special class of set functions that satisfy the property of diminishing returns. Defined over a finite ground set V, a set function $f : 2^V \to \mathbb{R}_+$ is said to be *submodular* [34], if for any $a \in V$ and subsets $A \subseteq B \subseteq V$, f satisfies the follows:

$$f(\{a\} \cup A) - f(A) \geq f(\{a\} \cup B) - f(B). \tag{1.1}$$

Many submodular functions can be optimized with theoretical guarantees using simple algorithms, which are also scalable for large data sets. A more detailed introduction of submodular optimization can be found is Chapter 2.

1.1 Robust Submodular Partitioning

The robust submodular partitioning problem [39], or often called "submodular fair allocation with indivisible goods", aims to find the partition such that the minimum-valued block in the partition is maximized according to the submodular function. The robust objective optimizes the worst block in the partition so that all blocks are minimally "good." The robust submodular partitioning problem has many applications. Given V as the training data for a machine learning task, we can find a partition of V for distributed training: every block of partitioned data is sent to a single machine for gradient computations in parallel, and the gradients are aggregated over all the blocks in the partition for model updates. Since we enforce each block to be representative of V, the gradients computed across distributed machines are consistent, resulting in reduced variance and improved convergence for the aggregation step. Using a similar idea, we can partition the training data into mini-batches so that every mini-batch is as representative as possible, therefore reducing the variance during gradient-based training.

We explore two different algorithmic approaches, *Min-Block Greedy* and *Round-Robin Greedy*, for our partitioning problem but under various constraints. For Min-Block Greedy based algorithms, we first show a $\frac{1}{m}$ bound for the unconstrained case and prove the bound is tight. We then modify the algorithm to allow a general down-closed constraint $\mathcal{C}$, and prove an approximation bound of $\frac{\alpha}{\alpha m+1}$, where α is the bound for solving the submodular maximization problem under constraint $\mathcal{C}$ using a greedy based algorithm. For Round-Robin Greedy based algorithms, when $\mathcal{C}$ is a cardinality constraint, we get a bound of $\frac{(1-e^{-1})^2}{3}$, and when $\mathcal{C}$ is a matroid constraint, we get a bound of $\frac{1-e^{-1}}{5}$. The Min-Block Greedy approach gives a weaker bound, but has a better running time $\mathcal{O}(n^2)$. The Round-Robin Greedy approach gives a constant bound, but its running time is worse: $\mathcal{O}(n^2(\log\log m + \log\frac{1}{\delta}))$, as it needs to binary search the optimal solution value to the given problem in an exponentially decreasing sequence, with $\frac{1}{1+\delta}(\delta > 0)$ as the decreasing factor (we assume an oracle model, and the running time is in terms of the number of submodular evaluations).

We further study the cardinality constrained robust partitioning problem and apply it to generate representative and fixed mini-batches for stochastic gradient methods. While sampling independent random mini-batches is essential from a theoretical perspective, it intrinsically conflicts with the efficient use of computational learning systems. Training sets are getting larger (thereby driving accuracy higher) and they typically do not fit in cache or memory. The only feasible approach is to repeatedly load data from main memory and/or disk to form mini-batches, but doing so from a convergence rate perspective (i.e., randomly with replacement) is costly because caches do not help when memory access patterns are random and hence unpredictable. As sequential access is significantly faster than random access, the only practical strategy is to iterate through a fixed sequence of data samples (written a priori to disk) rather than obtain an unbiased random mini-batch at every gradient update step.

Ideally, a good deterministic sequence of mini-batches should have the following properties: (1) mini-batch representativeness, where every mini-batch is representative and non-redundant, and thus stochastic gradients calculated using the mini-batches are not too far from the true gradients; and (2) order consistency, where groups of mini-batches in the sequence should be more broadly representative, so the corresponding sequence of gradients does not drive a model in the wrong direction. As mentioned above, however, one widely used approach to generate the fixed sequence is random shuffling, which could result in a suboptimal sequence due to non-representativeness. This can impede the performance of the trained model.

We propose to use cardinality constrained submodular robust partitioning to generate a sequence of mini-batches given a set of data samples. We prove a slight improved bound compared to the bound for general constraints using the min-block greedy algorithm. We also extend the partitioning to utilize a hierarchical structure. Our method consists of hierarchical runs of max-min robust submodular partition of the dataset with various cardinality constraints, which generates a partial order over mini-batches of data points within a hierarchical structure, and where both mini-batches and groups of mini-batches in the hierarchy

are encouraged to be representative of the entire dataset. On deep learning tasks, we show that our deterministic sequences of mini-batches significantly outperform the worst and mean of randomly generated sequences (both are likely to happen in practice), and for most of the cases, beat the best of random sequences.

1.2 Road Map of Part I

In Chapter 2, we introduce the preliminaries about submodular optimization that are relevant to this thesis.

In Chapter 3, we describe the robust submodular partitioning approach and propose algorithms with theoretical guarantees.

In Chapter 4 We apply the partitioning approach to generating fixed mini-batches for stochastic gradient methods to get improved performance and efficiency.

Chapter 3 was originally presented in three papers:

- K Wei, RK Iyer, S Wang, W Bai, JA Bilmes. *Mixed Robust/Average Submodular Partitioning: Fast Algorithms, Guarantees, and Applications*, In NeurIPS 2015.

- S Wang, W Bai, C Lavania, J Bilmes. *Fixing Mini-batch Sequences with Hierarchical Robust Partitioning*, In AISTATS 2019.

- S Wang*, T Zhou, C Lavania, J Bilmes.*Constrained Robust Submodular Partitioning*, Under Submission.

Chapter 4 was originally presented in the following paper:

- S Wang, W Bai, C Lavania, J Bilmes. *Fixing Mini-batch Sequences with Hierarchical Robust Partitioning*, In AISTATS 2019.

Chapter 2
PRELIMINARIES: SUBMODULAR OPTIMIZATION

This chapter introduces the background of submodular optimization that is most relevant to the first part of the thesis. Particularly, we will introduce the definition of submodular functions, matroids, and algorithms for submodular maximization. The materials introduced here are classic submodular optimization results, which come from the well-written books and surveys [34, 136, 55].

2.1 Submodular Functions: Definition and Examples

Submodularity is a natural property that arises in many combinatorial problems across fields of mathematics, economics, and operations research. Recently, submodularity has been widely applied to many machine learning tasks including data summarization, active learning and curriculum learning [151, 48, 107, 97, 66, 27, 60, 36, 22, 100, 127, 116, 109, 78, 138, 77, 76, 139, 153, 154, 133, 73].

2.1.1 Submodular Function Definition

Submodular function is a special class of set functions. A set function is defined over a ground set V of n items, and assigns a real value to any subset $A \subseteq V$. Formally, a set function is defined as $f : 2^V \to \mathbb{R}$. Since there are $2^{|V|} = 2^n$ subsets of V, an arbitrary set function has a lot of freedom in choosing its values and it is hard to describe or optimize such a set function without any constraints. Submodularity is a simple, general and natural constraint on a set function, which makes it possible to describe various properties and optimize problems involving set functions with submodularity. Also, submodularity arises in many real-world settings and submodular functions can be used to model many real-world tasks.

A submodular function $f : 2^V \rightarrow \mathbb{R}$ is a special class of set functions that satisfy the *diminishing returns* property defined as follows:

$$f(\{a\} \cup A) - f(A) \geq f(\{a\} \cup B) - f(B), \tag{2.1}$$

for any $A \subseteq B \subseteq V, a \in V, a \notin B$.

For simplicity of notations, we use $f(a|A)$ to denote the marginal gain of adding an item a to a set $A \subset V$. Precisely, we have $f(a|A) \triangleq f(\{a\} \cup A) - f(A)$. Intuitively, the definition of a submodular function implies that the gain of adding an element diminishes as the conditioning set becomes larger. Such diminishing return property arises in many real-world scenarios. For example, in economics, the law of diminishing marginal utility describes that the first unit of consumption of a good or service yields more utility than the subsequent utility.

Another definition of submodularity, which is also widely used in literature, is given as follows:

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B), \tag{2.2}$$

for any two subsets $A, B \subseteq V$. Note that the above definition and the diminishing return definition of a submodular function are equivalent. There are also many other equivalent definitions of submodular functions as shown in the following proposition:

Proposition 1 ([91])**.** *Each of the following statements is equivalent and defines a submodular function.*

- $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$, *for any* $A, B \subseteq V$.
- $f(a|A) \geq f(a|B)$ *for any* $a \in V$ *and* $A \subseteq B \subseteq V \setminus \{a\}$.
- $f(a|A) \geq f(a|A \cup a')$, *for any* $A \subset V$, $a \in V \setminus A$, *and* $a' \in V \setminus \{a\}$.
- $f(C|A) \geq f(C|B)$, *for any* $A \subseteq B \subseteq V$ *and any* $C \subseteq V \setminus B$.

- $f(B) \leq f(A) + \sum_{b \in B \setminus A} f(b|A)$, *for any* $A \subseteq B \subseteq V$.
- $f(A) \leq f(B) + \sum_{a \in A \setminus B} f(a|B) - \sum_{b \in B \setminus A} f(b|A \cup B \setminus b)$, *for any* $A, B \subseteq V$.

2.1.2 Supermodular and Modular Functions

A supermodular function can be thought as the "reverse" of a submodular function. Precisely, a function $h(\cdot)$ is a supermodular function if and only if $-h(\cdot)$ is a submodular function. We can also define supermodular functions through the "reverse" of the definitions given above about submodular functions by changing the inequality signs. Particularly, the most intuitive one is about increasing gains:

$$h(a|A) \leq h(a|B), \forall A \subseteq B \subseteq V, a \in V, a \notin B. \tag{2.3}$$

Supermodularity describes a from on complementary relationship among the elements in the groundset. We can think that elements may synergize with each other so their combined gain is larger than the individual gains.

A set function $m(\cdot)$ is a modular function if it is both submodular and supermodular. From the diminishing gains and increasing gains definitions, we can see that the gain of an element of a modular function does not change based on the conditioning set. Thus, a modular function $m(\cdot)$ always has the following property:

$$m(A) = \mathcal{C} + \sum_{a \in A} m(a), \forall A \subseteq V. \tag{2.4}$$

Where $\mathcal{C}$ is some constant and $m(\emptyset) = \mathcal{C}$. A modular function has only $n + 1$ degrees of freedom, and can be exactly represented by the value of every singleton element and the constant C. Since modular functions are both submodular and supermodular, the addition of a modular function and a submodular(supermodular) function preserves submodularity(supermodularity).

Unless stated otherwise, The submodular/supermodular and modular functions considered in this thesis are normalized and non-negative. For being normalized, the function f

should have $f(\emptyset) = 0$ and for being non-negative, $f(A) \geq 0 \; \forall A \subseteq V$. Another property that is also often required is the monotonicity, in which case $f(A) \geq f(B) \; \forall B \subseteq A$.

2.1.3 Examples of Submodular Functions

Submodular functions are a rich class of set functions that naturally describe the diversity property. Here we list a few very widely used submodular functions in the field of machine learning.

Facility Location Function

For a facility location function, the ground set V is considered as a set of locations, and we want to select a subset of the locations in V to build the facilities and measure the similarity (or negative distance) from any location in V to its closest facility in the selection. Formally, the facility location is defined as:

$$f(A) = \sum_{v \in V} \max_{a \in A} \text{sim}(a, v). \tag{2.5}$$

Here $\text{sim}(a, v)$ measures the similarity (or negative distance) between two locations a and v. The facility location function naturally describes the diversity of the subset A compared to the entire ground set V. A subset A with high $f(A)$ indicates that for every location $v \in V$, we can find some facility $a \in A$ that is very close/similar to v. Therefore, the subset A is representative of the ground set V. We will also see in following applications, by maximizing the facility location under certain constraints (e.g., cardinality), we can summarize or get a representative subset of items A from a large ground set V.

Even though the facility location function is very useful in practice, there are two major challenges when applying facility location functions. Firstly, the computation complexity of the facility location function is $O(n^2)$, so for cases where ground set size n is very large, the computational costs can be prohibitive. Moreover, it is non-trivial to et the right similarity measurement $\text{sim}(\cdot, \cdot)$ for the given dataset. Different choices of the similarity measurement may give dramatically different performance. In practice, we often utilize the representations

learnt using deep neural networks through some unsupervised or self-supervised learning procedure, and compute the similarity using a kernel, such as the RBF kernel. However, it usually takes effort to tune the parameters to get the best performing similarity measurement.

Feature Based Function

Given every element $v \in V$ with some feature representations $x(v) \in \mathbb{R}^m_{\geq 0}$, i.e., an $m-$dimensional non-negative vector for every v, we can define a feature based submodular function as follows:

$$f(A) = \sum_{j=1:m} \phi(\sum_{a \in A} x(a)[i]). \tag{2.6}$$

Where $x(a)[i]$ is the ith dimension of the $x(a)$ feature vector and ϕ is any concave function that is monotone (e.g., $\log(1+x)$, $\sqrt{x}$, and etc.). The feature based submodular function is indeed a concave over modular function. In general, for any non-negative modular function and a monotone concave function, the concave over modular composition is a submodular function.

The concavity of ϕ enforces the diminishing return property of added feature values. Therefore, to get a high function value, the selected subset should have high feature values on all dimensions, which enforces the diversity. In practice, we can use the one-hot representation of the features to enforce the diversity in a more intuitive way. For categorical features, say there are c different classes for this feature, then we can create a c length one-hot vector as the feature representation. By doing so, to maximize the feature based submodular function, we need to select a subset of items that have balanced number of classes for such feature. For numerical features, we can first cluster the values into categories and then apply the one-hot representation trick to get balanced items for each cluster of feature values.

Compared to the facility location function, the feature based submodular function has only $O(n)$ computational complexity. However, for numerical features, it requires extra efforts to do the clustering as mentioned above. In general, to design/process the right features for the function is non-trivial, and we cannot directly utilize the neural network representations as done in the facility location function case.

2.2 Matroid

A matroid is a combinatorial structure that is tightly connected with submodularity and submodular optimization. As we will discuss below, the matroid rank function is a submodular function. Also, many submodular optimization problems naturally have matroids as the constraints on the selected subsets (e.g., a partition constraint and a spanning-tree constraint). Submodular maximization under matroid constraints can often be solved with constant guarantees using greedy-like algorithms.

A matroid $\mathcal{M}(V, \mathcal{I})$ is formally defined on a finite ground set V, and an independence set system $\mathcal{I}$, where each member of $\mathcal{I}$ is a subset of V that is considered to have an independent property. Formally, the set of independent subsets $\mathcal{I}$ should satisfy the following three properties:

2.2.1 Definition

1. $\emptyset \in \mathcal{I}$,

2. Given $A \in \mathcal{I}$, $B \in \mathcal{I}, \forall B \subset A$.

3. $\forall A, B \in \mathcal{I}$ with $|A| > |B|$, there exists $a \in A \setminus B$ such that $B \cup \{a\} \in \mathcal{I}$.

The first property requires that the empty set $\emptyset$ is always considered an independent set. The second property is also known as the down-closed property, which indicates that any subset of an independent set is also an independent set. The third property is also noted as the exchange property: for any independent sets A and B with $|A| > |B|$, we can find at least one item $a \in A \setminus B$ such that $B \cup \{a\}$ is still an independent set.

Intuitively, a matroid describes some form of "independence" among the elements in the ground set. In the setting of real vectors, the independence can be the linear independence among a set of vectors. For graphs, the independence can be a set of edges that do not contain any cycles. We will discuss those in more details in the matroid examples below.

Given the exchange property, intuitively, the cardinality of the largest independent set of a matroid $\mathcal{M}$ should have the same size. Such inclusion-wise maximally independent set is also called a base of the matroid. Indeed, all the bases of the matroid has the same cardinality. The cardinality of the bases of a matroid is also referred as the rank of the matroid. We can extend the notion of rank to a subset $A \subseteq V$. Formally, the rank function of a matroid for the input set A is defined as:

$$r_{\mathcal{M}}(A) = \max\{|X| : X \subseteq A, X \in \mathcal{I}\} = \max_{X \in \mathcal{I}} |A \cap X|. \tag{2.7}$$

We note that $r(A) \leq |A|$, and the equality is satisfied only when A is an independent set according to the matroid, or in other words, $A \in \mathcal{I}$. The matroid rank function $r_{\mathcal{M}}$ is a submodular set function for any choice of the matroid $\mathcal{M}$. Intuitively, we can think the matroid rank function as a submodular function with gains equal to either 0 or 1. A monotone non-negative and normalized submodular function is also referred as a polymatroid function.

Matroids are also tightly connected with the greedy algorithm, which is a widely used approach to solve combinatorial optimization problems by adding the element one by one with the largest gain. In fact, the matroid definition above is shown to be equivalent to a property about solving the optimization problem using the greedy algorithm. Specifically, for the problem of maximizing a non-negative modular function under an independence set system constraint, we have:

Theorem 1. *[29] Let $\mathcal{I}$ be an independence system. The $(V, \mathcal{I})$ is a matroid if and only if for every modular function $m \in \mathbb{R}_{\geq 0}$, Algorithm 1 gives a solution that optimizes Eqn 2.8 exactly.*

$$\max_{A \in \mathcal{I}} m(A), \tag{2.8}$$

where $m : V \to \mathbb{R}_+$ is a modular function.

Theorem 1 shows that the greedy algorithm is a defining characteristic of a matroid. However, we note that for solving more general problems such as optimizing a submodular

Algorithm 1: Greedy algorithm for modular maximization under a set system constraint.

Input: A modular function $m : V \to \mathbb{R}_{\geq 0}$ and a set system constraint $(V, \mathcal{I})$.
$A := \emptyset$;
$R := V$;
while $\exists a \in R : A \cup \{a\} \in \mathcal{I}$ **do**
    $a \in \operatorname{argmax}_{a \in R : A \cup \{a\} \in \mathcal{I}} m(a)$;
    $A := A \cup \{a\}$;
    $R := R \setminus \{a\}$;
Output: A.

function under a matroid constraint, the greedy algorithm is not guaranteed to give the optimal solution.

2.2.2 Matroid examples

Linear Matroid

For the setting of real vectors, we can describe the linear independence of vectors with the independence system of a matroid. Suppose we are given a ground set V of n real vectors, where each vector has d dimensions. The independence sets of a linear matroid contain all subsets of vectors in V that are linearly independent. Therefore, the rank function $r(A)$ gives the rank of the vector space spanned by the vectors in A. Since the vectors are of d dimensions, the maximum rank is d for any subset of V. As the linear matroid is one specific kind of a matroid, matroids generalize the linear-independence.

k-Uniform Matroid

As a constraint, the k-uniform matroid is the same as a k cardinality constraint. Formally, the independent sets of the k-uniform matroid contain all subsets of cardinality at most k, i.e., we have:

$$\mathcal{I} = \{A \subseteq V : |A| \leq k\}. \tag{2.9}$$

Therefore, the rank function is:

$$r(A) = \min\{|A|, k\}. \tag{2.10}$$

It is easy to verify the down-closed property and the exchange property as any subset of size at most k is considered independent. We can also verify the submodularity of the rank function as it is essentially a monotone concave function $\min(\cdot, k)$ over a non-negative modular function $|A|$. Since the k-uniform matroid constraint is the same as a k cardinality constraint, and the k-uniform matroid is one specific kind of a matroid, the matroid constraints generalize cardinality constraints for optimization problems.

Partition Matroid

A partition matroid can be seen as a generalization of the k-uniform matroid, where we don't treat all elements of V "uniformly". Formally, the independent sets of a partition matroid is defined as follows:

$$\mathcal{I} = \{A \subseteq V : |A \cap V_i| \leq k_i, \forall i = 1, \ldots, m\}. \tag{2.11}$$

Where $V = \{V_1, \ldots, V_m\}$, and $V_i \cap V_j = \emptyset$, i.e., $V_1, \ldots, V_m$ form a partition of the groundset V of m blocks. In the special case where $m = 1$ and $V_i = V$, $k_i = k$, we recover the k-uniform matroid. When we have $m > 1$, we can think that given a predefined partitioning of the ground set, we constrain the cardinality of elements selected from each of the partitioned block. The rank function of a partition matroid is given by:

$$r(A) = \sum_{i=1}^{m} \min\{|A \cap V_i|, k_i\}. \tag{2.12}$$

The partition matroid can be applied to constrain the solution set to have some balanced property with respect to certain predefined partitioning. For example, in the case of selecting a subset of training samples for classification tasks, the class labels form natural partitions of the ground set, and we can use the partition matroid to balance the samples selected from each class.

Cycle Matroid

Given a graph and let the ground set V contain the edges of the graph, the independence sets of a cycle matroid contain all subsets of edges that do not form cycles based on the graph structure. Therefore, the rank of the matroid is equal to the size of the maximally independent forests of the subset of edges. Given optimization problems with natural graphical structures, we can use the cycle matroid to constrain the solution to have no cycles. For example, given a network of devices, with the cycle matroid, we can find a subset of connections to propagate the information in a non-cyclical manner to improve the communication efficiency.

2.3 Continuous extensions of submodular functions

One challenge for submodular optimization is that a submodular function is a discrete function with 2^n degrees of freedom. To address the challenge, various continuous extensions of a submodular functions have been proposed. Given the continuous extensions, the optimization problems can typically be solved in a two step manner: 1) solve the continuous problem using gradient based methods; 2) round the continuous problem solution, which has fractional values, to an integral solution that can be represented by sets.

The advantages of such a approach include: 1) we can benefit from the rich literature of continuous optimization methods; 2) for solving new problems with similar structures and properties, it is often easier to extend the existing continuous methods, or in other words, the continuous approach mentioned above is a general approach that can be applied to many different problems, while to solve the combinatorial problem directly usually result in dra-

matically different algorithms for different problems. However, there are also some major drawbacks: 1) it is nontrivial to know whether the continuous algorithm converges, and the implementation needs to be careful about the numerical problems; 2) the continuous extensions are typically very expensive to evaluate, making such approaches mostly interesting in the realm of theoretical analysis.

We will describe two widely used extensions of submodular functions below: the Lovász extension, which is often used for solving submodular minimization and the multilinear extension which is often used for solving submodular maximization.

2.3.1 Lovász extension

Even though the first part of the dissertation is more related to submodular maximization, the Lovász extension, while being mostly useful for submodular minimization, reveals important properties of submodular functions as described in Theorem 2 below. For a set function $f : 2^V \to \mathbb{R}$, which can be thought as a function defined on hypercubes of $\{0,1\}^n$, the Lovász extension [79] of the set function extends the domain of the function to $[0,1]^n$: $\tilde{f} : [0,1]^n \to \mathbb{R}$, and matching the function values on the hypercubes. The Lovász extension is defined as follows:

Definition 1. *For any $x \in [0,1]^n$, let σ_x be an ordering its elements in a non-increasing order, i.e., $x[\sigma_x[i]] \geq x[\sigma_x[j]] \forall i < j$. We can then form a chain of subsets based on the ordering: $S_0^{\sigma_x} \subset, \ldots, \subset S_n^{\sigma_x}$ where $S_i^{\sigma_x} = \{\sigma_x[1], \ldots, \sigma_x[i]\}$ for $i = 1, \ldots, n$, i.e., the first i elements in the ordering σ_x. For a set function f, its Lovász extension $\tilde{f}$ is defined as:*

$$\tilde{f}(x) = \sum_{i=1}^{n} x(\sigma_x(i))(f(S_i^{\sigma_x}) - f(S_{i-1}^{\sigma_x})). \tag{2.13}$$

For function values on the hypercube, we can check that f and $\tilde{f}$ have exactly the same values as $\tilde{f}(x)$ essentially accumulates the function value based on a certain order of the elements in the set when x is on the hypercube. Also note that $\tilde{f}$ is piece-wise linear. The following theorem shows the equivalence of the Lovász extension being convex and the set function f being submodular.

Theorem 2 ([79]). *For a set function $f : 2^V \to \mathbb{R}$ and its Lovász extension $\tilde{f} : [0,1]^n \to \mathbb{R}$, f is submodular if and only if $\tilde{f}$ is convex.*

As mentioned above, for submodular minimization, instead of solving the combinatorial problem directly, we can solve the convex problem introduced in the Lovász extension, and then round the fractional solution to an integral solution on the hypercube to get the solution set. We can therefore benefit from the rich literature of convex optimization techniques [12] such as gradient based methods or ellipsoid methods to solve the Lovász extension problem.

2.3.2 Multi-linear extension

Similar to the case of Lovász extension, given a set function $f : 2^V \to \mathbb{R}$, we can extend its domain to $[0,1]^n$ by using introducing a multi-linear extension [130] $F : [0,1]^n \to \mathbb{R}$ as follows:

Definition 2. *For a set function $f : 2^V \to \mathbb{R}$, its multi-linear extension be $F : [0,1]^n \to \mathbb{R}$ is defined as:*

$$F(x) = \sum_{A \subseteq V} f(A) \prod_{a \in A} x_a \prod_{a \in V \setminus A} (1 - x_a). \tag{2.14}$$

Here we can think the fractional input argument x as a probability distribution to sample elements in the ground set V and the multi-linear extension returns the expected value of the sampled subsets. More precisely, every element v_i is sampled independently based on the fractional value in the ith dimension of x. We can also denote the multi-linear extension as $F(x) = \mathbb{E}_{R \sim x} f(R)$. Therefore, it is also easy to check that the multi-linear extension matches the function evaluations on the hypercubes as in such a case the elements are sampled with probability 1. Although evaluating $F(x)$ for general x requires an exponential number of queries on the function f, concentrated estimate of the multi-linear extension $F(\cdot)$ can be achieved using a polynomial number of evaluation of the function f on random subsets based on the Chernoff bounds [130].

Even when f is submodular, the multi-linear extension is not a concave function. However, we have the following concave-like properties of the multi-linear extension when f is submodular. Similar to the Lovász extension case, the following properties and the submodularity of f are equivalent:

Theorem 3. *[16] For a set function $f : 2^V \rightarrow \mathbb{R}$, let its multi-linear extension be $F : [0,1]^n \rightarrow \mathbb{R}$. f is submodular if and only if F satisfies the following:*

$$\frac{\partial^2 F(x)}{\partial x_i \partial x_j} \leq 0, \forall i, j, x \in [0,1]^n. \tag{2.15}$$

The multi-linear extension is particularly useful for solving submodular maximization problems. [130] proposes a continuous greedy algorithm on the multi-linear extension objective, which gives a guarantee matching the theoretical hardness when solving submodular maximization under a matroid constraint. For the later part of the dissertation, where we target the problem of constrained robust submodular partitioning, we also use properties of the continuous greedy and multi-linear extension to address the problem. However, we also note that the multi-linear extension is not practically very useful due to its high costs of evaluation. For the continuous greedy algorithm, to get high probability bounds, $O(n^5)$ subset samples based on x need to be made which are typically prohibitive in practice.

2.4 Submodular Function Optimization and Existing Algorithms

2.4.1 Submodular Maximization

Submodular maximization generalizes many known NP-hard problems, such as max graph cut and set cover. Instead of solving the problem exactly, we aim to find efficient approximation algorithms that are proven to have approximation ratios compared to the optimal solution.

Monotone Submodular Maximization

Here we discuss submodular maximization for monotone submodular functions. Since the function is monotone, the problem is only non-trivial with a constraint as otherwise we

can just select the entire ground set. Firstly, we aim to solve the problem of submodular maximization under a cardinality constraint, formally defined in Problem 1. Due to the diminishing return property, the monotone submodular function naturally describes the representativeness or the diversity of the input set argument. By optimizing the submodular function under a cardinality constraint, we want to find a set of a limited size that is most representative based on the submodular evaluation. Such optimization problem arises naturally in many real world situations, such as text summarization and training data selection.

Problem 1 (Cardinality Constrained Submodular Maximization)**.**

$$\max_{A \subseteq V, |A| \leq k} f(A) \tag{2.16}$$

Algorithm 2: Greedy algorithm for Problem 1 [91]

Input: k, V, and f.
Initialization: $A := \emptyset$, $R := V$;
for $i = 1$ *to* k **do**
    Let $a^* \in \text{argmax}_{a \in R} f(a|A)$;
    $A := A \cup a^*$;
    $R := R \setminus a^*$;
return A

Theorem 4. *[91] Algorithm 2 is guaranteed to output a solution S such that*

$$f(S) \geq (1 - 1/e) f(O) \approx 0.63 f(O), \tag{2.17}$$

where $O \in \text{argmax}_{|A| \leq k, A \subseteq V} f(A)$ is the optimal solution for Problem 1.

Theorem 5 (Theoretical Hardness of Cardinality Monotone Submodular Max [30])**.** *Unless $P = NP$, there does not exist any polynomial time algorithm that solves Problem 1 with an approximation factor better than $(1 - 1/e)$.*

The greedy algorithm (Alg. 2) has k iterations, and at every iteration, it selects the next element with the highest gain conditioned on the already selected elements. The time complexity of the greedy algorithm is therefore $O(nk)$. When k is on the same scale as n, the complexity becomes $O(n^2)$. Note that we discuss about the time complexity in the oracle model, in which case we assumes that we can access an oracle of the submodular function to get its evaluations and the complexity is based on the number of times we call the submodular function oracle. In practice, the submodular function can be more expensive to evaluate (e.g., the facility location function evaluation takes $O(n^2)$ complexity). The greedy algorithm has an approximation ratio of $1 - e^{-1}$, which is also shown to be the best possible approximation ratio for the problem of submodular maximization under a cardinality constraint using a polynomial time algorithm unless $P = NP$.

The quadratic complexity of the greedy algorithm can be quite costly or even infeasible for very large ground set sizes. Note that at every iteration of the greedy algorithm, we aim to find an element with the highest gain, and we can utilize the diminishing return property of the submodular function to reduce the number of elements searched to find the element with the highest gain. Such acceleration method is also known as the lazy greedy algorithm [80]. For lazy greedy, we maintain a priority queue storing the marginal gains of every element conditioned on some subset (starting with the empty set). At iteration t of the greedy algorithm, we pop the element a_1 with the highest gain from the priority queue and evaluate its true gain conditioned on the currently selected elements A_{t-1}. If the gain is larger than the next best element in the priority queue, then based on the submodularity we can conclude that a_1 is indeed the element with the highest gain conditioned on A_{t-1} as $f(a_1|A_{t-1}) \geq f(a|A') \geq f(a|A_{t-1})$ for all $a \in V \setminus A_{t-1}$. Otherwise, if a_1 is not better than the next gain, we just push $f(a_1|A_{t-1})$ into the priority queue and repeat. In the worst case, we still need to go over the entire priority queue to find the element with the highest gain, so the worst case time complexity is still $O(n^2)$. In practice, we find this trick very powerful. As we will discuss later in the robust partitioning problem, we do an empirical study showing that the lazy greedy trick reduces the time complexity to around $O(n^{1.3})$.

Another line of research to reduce the complexity of the greedy algorithm aims to find an element that has a large enough gain rather than the highest gain as done in the greedy algorithm. [6] propose a thresholding based greedy algorithm, which gives an approximation ratio of $(1 - 1/e - \epsilon)$ with $O(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$ function valuations. Here, ϵ is an extra parameter for the algorithm, where we can get better approximation guarantee at the cost of higher running time complexity. Essentially the algorithm maintains a threshold that gets decreased exponentially, and at every iteration, we add in all elements that have gains larger than the current threshold.

Mirzasoleiman et al. [81] give a lazier-than-lazy greedy algorithm, which achieves the same approximation ratio of $(1 - 1/e - \epsilon)$ with $O(n \log \frac{1}{\epsilon})$ function calls. Similar to the previous case, we can trade off the approximation ratio with the running time complexity by tunning the ϵ parameter. The lazier-than-lazy greedy algorithm has k iterations, and at every iteration it first uniformly samples a subset from the ground set, and then finds the element of the highest gain from the sampled subset. The ϵ parameter controls the size of the sampled subset. As also claimed in the paper [81], it is possible to integrate the lazy-greedy trick for this algorithm. However, it requires an efficient implementation of sampling/updating of the priority queue data structure (it is not fully solved and for functions that do not have high evaluation costs, such as feature based functions, the sampling operations from the priority queue may have a higher computational complexity, which defeats the purpose of the lazy-greedy trick).

As discussed in the matroid section above, the cardinality constraint is equivalent to a k-uniform matroid constraint. For the following part, we extend the cardinality constraint to the matroid constraint case, and more generally, an intersection of p matroids. The intersection of p matroids constrains that the selected set should satisfy every one of the p matroids. Also note that an intersection of p matroids is generally not a matroid.

Problem 2 (Matroid constrained submodular maximization)**.**

$$\max_{A \in \bigcap_{j=1:p} \mathcal{I}_p(\mathcal{M}_p)} f(A), \tag{2.18}$$

where $\mathcal{M}_{|}(V, \mathcal{I}_{|})$ is a matroid.

Algorithm 3: Greedy algorithm for submodular maximization under a matroid constraint.

Input: A submodular function $f : V \to \mathbb{R}_{\geq 0}$ and p matroids $\mathcal{M}_j(V, \mathcal{I}_j)$.
Initialize: $A := \emptyset$, $R := V$;
while $\exists a \in R : A \cup \{a\} \in \bigcap_{j=1:p} \mathcal{I}_j$ **do**
    $a \in \operatorname{argmax}_{a \in R : A \cup \{a\} \in \bigcap_{j=1:p} \mathcal{I}_j} f(a|A)$;
    $A := A \cup \{a\}$;
    $R := R \setminus \{a\}$
return A

Theorem 6 ([33]). *Algorithm 3 is guaranteed to find a solution S such that*

$$f(S) \geq \frac{1}{p+1} f(O), \tag{2.19}$$

where O is the optimal solution to Eq. 2, i.e., $O \in \operatorname{argmax}_{A \in \bigcap_{j=1:p} \mathcal{I}_j(\mathcal{M}_j)} f(A)$.

A similar greedy algorithm (Alg. 3) gives a $\frac{1}{p+1}$ approximation guarantee for the problem of submodular maximization under an intersection of p matroid constraints. At every iteration of the greedy algorithm, it finds the element with the highest gain that also satisfies the constraints. When there is only one matroid constraint, the approximation guarantee therefore becomes 1/2. The approximation ratio is also shown to be tight for the greedy algorithm. The running time complexity is still $O(n^2)$ in the worst case, and we can also use the lazy-greedy trick as described for the problem of submodular maximization under a cardinality constraint to accelerate the running time in practice.

In the case of a single matroid constraint ($p = 1$), the 1/2 approximation ratio for Eq. 2 turns out to be sub-optimal for the hardness of $1 - e^{-1}$. Vondrák [130] propose a continuous greedy algorithm, which gives an approximation ratio of $1 - e^{-1}$ that matches the theoretical

hardness. The approach lies in the framework of first solving a continuous extension of the submodular problem and then rounding the fractional solution to a set. The algorithm utilizes the multi-linear extension of the submodular function, and keeps a fractional solution. At every iteration, we find the direction that gives the fastest increase in the function value conditioned on the current fractional solution for the multi-linear extension evaluation, and we move in that direction for a certain step size. In the end, we get a fractional solution that has a value of at least $1 - e^{-1}$ of the optimal solution, and a pipage-rounding is applied to round the fractional solution to an integral solution with no loss of function value.

The continuous greedy algorithm also reveals an interesting property for the problem of submodular maximization under a partition matroid constraint, which is also known as the homogeneous submodular welfare problem. It is shown that the continuous greedy algorithm is equivalent to uniformly distributing elements into blocks of the partition matroid, and therefore, the random assignment strategy matches the theoretical hardness for such a problem. This also works as one key insight in our following studies of the robust submodular partitioning problem.

As mentioned before, the multi-linear extension of a submodular function is extremely costly to evaluate computationally and a naive implementation takes $O(n^5)$ samples to approximate the multi-linear extension function value. As a result, the continuous greedy takes around $O(n^7)$ calls to the submodular function oracle, which is not practically applicable.

Another constraint that generalizes the cardinality constraint is the knapsack constraint. For a knapsack constraint, we have a non-negative modular function c, which denotes the cost of every element. We also have a budget b, and we aim to find a subset A from the ground set such that the sum of the costs is less than the budget: $c(A) \leq b$. Note that the knapsack constraint is not a matroid constraint, as we may have maximal subsets for the knapsack constraint that do not have the same cardinality. In the case where $c(v) = 1$ and $b = k$, the knapsack constraint recovers the cardinality constraint.

Problem 3 (Knapsack constrained submodular maximization)**.**

$$\max_{A\subseteq V, c(A)\leq b} f(A), \tag{2.20}$$

where $c : V \to \mathbb{R}$ is a modular function measuring the cost of each item $a \in V$, and b is the budget.

Algorithm 4: Greedy algorithm for Problem 3 [65]

Input: b, V, c, and f.
Initialization: $A := \emptyset$, R:=V;
while $c(A) < b$ **do**
    Let $a^* \in \text{argmax}_{a\in R: c(A\cup\{a\})\leq b} \frac{f(a|A)}{c(a)}$;
    $A := A \cup a^*$;
Let $a^* \in \text{argmax}_{a\in V: c(a)\leq b} f(a)$;
return $\text{argmax}_{A'\in\{A,\{a^*\}\}} f(A')$

When the function f in Problem 3 is modular, the problem becomes the well-known knapsack problem. The greedy algorithm (Alg. 4) approaches the problem in a similar manner as previous greedy algorithms but has two modifications: 1) at every iteration, it finds the element with the highest normalized gain $\frac{f(a|A)}{c(a)}$ as opposed to the gain; 2) in the end, it compares the solution to the best singleton element in the remaining elements. The algorithm is proven to have an approximation ratio of $\frac{1-e^{-1}}{2}$, as shown below:

Theorem 7 ([65])**.** *Algorithm 4 outputs a solution S such that*

$$f(S) \geq \frac{1-1/e}{2} f(O), \tag{2.21}$$

where O is the optimal solution to Eq. 3, i.e., $O \in \text{argmax}_{A\subseteq V: c(A)\leq b} f(A)$.

The worst case time complexity is $O(n^2)$, and we can also use the lazy-greedy trick to accelerate the algorithm in practice, and we need to store the normalized gain $\frac{f(a|A)}{c(a)}$

in the priority queue. The approximation ratio does not match the theoretical hardness. However, a simple change of the algorithm is shown to have an approximation ratio of $1 - e^{-1}$, and since the knapsack constraint generalizes the cardinality constraint, $1 - e^{-1}$ is also the theoretical hardness for the knapsack constrained problem. The simple change is known as the partial enumeration, where we first enumerate all possible solutions that satisfy the knapsack constraint with cardinality no more than three, and then based on every such possible solution, we run the greedy algorithm (Alg. 4), and return the best solution in the end. Because of the enumeration process, the time complexity becomes $O(n^5)$, which can be infeasible. In practice, it is often the case that the simple greedy algorithm can yield good solution to the problem of submodular maximization under a knapsack constraint.

Generalized forms of Problem 3 have also been studied in the literature. Kulik et al. [72] consider the problem of multiple knapsack constraint and gives an algorithm achieving an approximation ratio of $(1 - 1/e - \epsilon)$. Iyer and Bilmes [53] study the problem of submodular maximization under a submodular cost constraint, in which case the modular function $c(\cdot)$ mentioned above is replaced with a non-negative and monotone submodular function. The generalized problem is, however, much harder as it is shown to have an information theoretically hardness of $O(\frac{1}{\sqrt{n}})$. They also give an approximation algorithm with a guarantee of $O(\frac{1}{\sqrt{n}\log n})$ matching the lower bound up to a log factor. Badanidiyuru et al. [7] study the problem of maximizing a monotone submodular function under an intersection of p matroid constraints and l knapsack constraints with $l \leq k$, and give an algorithm with an approximation factor of $2(k+1) + \epsilon$ with time complexity $O(nk \log(\frac{n}{\epsilon})) \log(\frac{1}{\epsilon})r$.

Problem 4 (Non-monotone Submodular Maximization)**.**

$$\max_{A \subseteq V} f(A) \tag{2.22}$$

For the case where the submodular function is non-monotone, the problem is non-trivial even without a constraint. Previous results show that it is NP-hard to check if the maximal value of a non-monotone submodular function is positive or negative, so for the non-monotone optimization problem, we restrict the function to be non-negative.

A simple approach of uniform random sample every element with a probability of 1/2 gives an approximation ratio of 1/4 in expectation. A bi-directional greedy algorithm [14] gives an approximation ratio of 1/3, and the randomized version of the bi-directional greedy (Alg. 5) gives an approximation ratio of 1/2 in expectation, which matches the theoretical hardness as given in [31].

Algorithm 5: Bi-directional greedy algorithm for Problem 4 [14]

Input: An arbitrarily ordered ground set $V = \{v_1, \ldots, v_n\}$.
Initialize: $X_0 := \emptyset$, $Y_0 := V$;
for $i = 1, \ldots, n$ **do**
  $a_i := \max\{0, f(v_i|X_{i-1})\}$;
  $b_i := \max\{0, -f(v_i|Y_{i-1} \setminus v_i)\}$;
  Sample a number ρ uniformly and independently at random from $[0, 1]$;
  if $\rho \leq \frac{a_i}{a_i+b_i}$ **then**
    $X_i := X_{i-1} \cup \{v_i\}$;
    $Y_i := Y_{i-1}$;
  $X_i := X_{i-1}$;
  $Y_i := Y_{i-1} \setminus \{v_i\}$;
return X_n (or equivalently Y_n)

We show the randomized version of the bi-directional greedy algorithm in Alg. 5. The algorithm maintains two sets X and Y, where X starts with $\emptyset$ and Y starts with the ground set. We pick an arbitrary order over the elements in the ground set, and at every iteration, we decide for the next element in the ordering whether to add the element to X or remove it from Y depending on the gain values. In the end, the elements in X are the same as the elements in Y, as if we add an element to X, that element is also in Y, and if we remove an element from Y, that element is not added to X. Intuitively, since X and Y meet in the

end, adding the element to X or removing from Y both can increase the final value, and we greedily pick the one that has the largest increase. The bi-directional greedy requires only $O(n)$ calls to the submodular function oracle and can operate in a streaming manner. [13] shows a de-randomization of the randomized version of the bi-directional greedy algorithm, that achieves a $1/2$ approximation ratio deterministically.

There are also many works on the constrained case of the non-monotone submodular maximization problem. For the cardinality constraint case, [15] give an algorithm with an approximation ratio of $1/e$. [71] propose an interlace greedy algorithm that give an approximation ratio of $1/4 - \epsilon$ with $O(\frac{n}{\epsilon}\log(\frac{k}{\epsilon}))$ complexity. For the case of an intersection of p matroid constraints, [75] show an approximation ratio of $\frac{1}{2+p+\frac{1}{p}}$. They also show an approximation ratio of $1/5$ for the case of l knapsack constraints. [131] study the case of a matroid base constraint and give an approximation algorithm with a factor of 0.309. [37] improve the factor to 0.325 with a simulated annealing approach, and [32] further improve it to $1/e$.

2.4.2 Submodular Minimization

Different from the submodular maximization problem, the submodular minimization problem often utilizes the convexity property of the Lovász extension of the submodular function. As we will discuss below, the unconstrained submodular minimization problem can be solved exactly in polynomial time, and for the constrained case, even with a cardinality constraint, the problem becomes much harder with a lower bound of $o(\sqrt{\frac{n}{\log n}})$.

Unconstrained case

For the unconstrained case, the problem is only non-trivial if the submodular function is non-monotone, as otherwise we can just select the empty set as the minimizer. Given that the Lovász extension of the submodular function is convex, many continuous optimization techniques are applied to solve the submodular minimization problem. Grötschel et al. [40] utilize the ellipsoid method to minimize the Lovász extension in strongly-polynomial time.

Fujishige et al. [35] propose a min-norm point algorithm to minimize the dual of Lovász extension using proximal methods, which is widely used in practice. However, the running time for the min-norm algorithm is not well understood, and Chakrabarty et al. [17] give a pseudo-polynomial time guarantee such that the min-norm algorithm can find the minimizer in $O((n^5\gamma + n^7)F^2)$ time with $F = \max_{a \in V}\{|f(a)|, |f(V) - f(V \setminus a)|\}$. Jiang [58] show an algorithm with $O(n^3)$ running time using convex geometry, which gives the best known running time algorithm for the submodular minimization problem.

Other than the continuous optimization approach, there are also combinatorial algorithms developed for the submodular minimization problem. Firstly, Cunningham [26] give a strongly-polynomial time algorithm framework for the membership problem for matroid polyhedra, which is a special case of the submodular minimization problem. Based on that, Iwata et al. [52] propose an algorithm with complexity $O(Cn^7 \log n)$, Schrijver [105] give an algorithm with complexity $O(n^8 + Cn^7)$, and Orlin [94] give an improved running time of $O(Cn^5 + n^6)$, where C is the time complexity of running one function evaluation oracle. Compared to the continuous approach, the combinatorial approach costs higher time complexity and is therefore not practically useful for real-world tasks. However, it's also worth mentioning that the continuous approach usually suffers from numerical problems that require careful human tunning, and if incorrect parameters are applied, the algorithm may not terminate at the true minimum.

Constrained case

Even with a cardinality constraint, the submodular minimization problem becomes much harder and Svitkina and Fleischer [123] show a lower bound that it is information theoretically hard to approximate the problem within a factor of $o(\sqrt{\frac{n}{\log n}})$. Nagano et al. [88] show that by using a similar min-norm point algorithm as used for the unconstrained case, we can potentially get solutions to the cardinality constrained submodular minimization problem for certain cardinality values, but we cannot decide on the values. Grötschel et al. [40] show that when the constraint becomes either odd or even sets, the problem can be solved

exactly in polynomial time. Nägele et al. [90] generalize that and give an algorithm for the submodular minimization under congruency constraints, in which case the selected subset size $|A| \equiv r \mod m$, where r and m are some parameters for the constraint. They show that when m is a prime member, the problem can be exactly solved in polynomial time. Iyer et al. [54] propose a majorization-minimization framework for solving the constrained submodular minimization under general constraints. The approach is an iterative method that solves a transformed version of the problem that replaces the submodular function to its tightest modular upper bound, and such modular optimization problem can be solved exactly and efficiently. The approximation guarantee for this approach depends on the curvature of the submodular function, which measures how curved the submodular function is, or intuitively how well can we approximate the submodular function as a modular function.

Chapter 3

THEORETICAL RESULTS ON SUBMODULAR ROBUST PARTITIONING

The problem of partitioning a given set V of items into m blocks, where any two blocks share no items in common, arises in many real-world scenarios and machine learning applications. As an optimization problem, partitioning aims to generate the blocks so that the utilities of the blocks, as measured by a given set function, are good. submodular utility function for each partitioned block, we encourage each block to be representative of the ground set V. Many algorithms have been proposed for various settings of submodular partitioning problems with approximation guarantees.

For the submodular welfare problem [130], we aim to find a partition such that the sum of the submodular evaluations of every block is maximized. Such an objective promotes the overall utility of the entire partition but some blocks may still have small function values. The robust submodular partitioning problem [39], or often called "submodular fair allocation with indivisible goods", aims to find the partition such that the minimum-valued block in the partition is maximized according to the submodular function. The robust objective optimizes the worst block in the partition so that all blocks are minimally "good." In the general setting, every block in the partition may have a different submodular function (the heterogeneous case) although for this work, we study only the restricted setting where all blocks share the same submodular function (the homogeneous case). The robust submodular partitioning problem has many applications. Given V as the training data for a machine learning task, we can find a partition of V for distributed training: every block of partitioned data is sent to a single machine for gradient computations in parallel, and the gradients are aggregated over all the blocks in the partition for model updates. Since we enforce each block

to be representative of V, the gradients computed across distributed machines are consistent, resulting in reduced variance and improved convergence for the aggregation step. Using a similar idea, we can partition the training data into mini-batches so that every mini-batch is as representative as possible, therefore reducing the variance during gradient-based training.

In this work, we explore two different algorithmic approaches, *Min-Block Greedy* and *Round-Robin Greedy*, for our partitioning problem but under various constraints. For Min-Block Greedy based algorithms, we first show a $\frac{1}{m}$ bound for the unconstrained case and prove the bound is tight. We then modify the algorithm to allow a general down-closed constraint $\mathcal{C}$, and prove an approximation bound of $\frac{\alpha}{\alpha m+1}$, where α is the bound for solving the submodular maximization problem under constraint $\mathcal{C}$ using a greedy based algorithm. For example, for a cardinality constraint, $\alpha = 1 - 1/e$ [29], and the bound for the constrained robust submodular partition is $\frac{1}{m+\frac{1}{1-1/e}}$. Similarly, for $\mathcal{C}$ as an intersection-of-p-matroids constraint, $\alpha = \frac{1}{1+p}$ [33], and the bound is $\frac{1}{m+p+1}$; for $\mathcal{C}$ as a knapsack constraint, $\alpha = 0.5(1 - 1/e)$ [65], and the bound is $\frac{1}{m+\frac{2}{1-1/e}}$. For the cardinality constraint case in particular, we can prove a slightly improved bound of $\frac{e-1}{(e-1)m+1}$. Moreover, we also propose a hierarchical version of the Min-BLogk Greedy algorithm for the cardinality constraint case, and show an approximation guarantee depending on some mild assumptions about the partitioning process. For Round-Robin Greedy based algorithms, when $\mathcal{C}$ is a cardinality constraint, we get a bound of $\frac{(1-1/e)^2}{3}$, and when $\mathcal{C}$ is a matroid constraint, we get a bound of $\frac{1-1/e}{5}$. The Min-Block Greedy approach gives a weaker bound, and since the $\frac{1}{m}$ bound for the unconstrained is tight, we cannot improve upon the $\frac{1}{m}$ factor for the constrained case. The Round-Robin Greedy approach gives a constant bound, but its running time is worse. The running time for Min-Block Greedy is $\mathcal{O}(n^2)$, where n is the ground set size. For Round-Robin Greedy under a matroid constraint, the running time is $\mathcal{O}(n^2(\log\log m + \log\frac{1}{\delta}))$, as it needs to binary search the optimal solution value to the given problem in an exponentially decreasing sequence, with $\frac{1}{1+\delta}(\delta > 0)$ as the decreasing factor (we assume an oracle model, and the running time is in terms of the number of submodular evaluations). Importantly, by utilizing the Min-Block Greedy algorithm result first, our Round-Robin Greedy algorithm can have

a strongly polynomial running time, while previous results on the unconstrained case using a Round-Robin like algorithm only has weakly polynomial running time [9]. We summarize our theoretical results in the following Table 3.1.

Table 3.1: Theoretical Results on the Robust Submodular Partitioning Problem

Constraint	Algorithm	Approx. Ratio	Running Time
None	Min-Block Greedy (Alg. 7)	$1/m$	$\mathcal{O}(n^2)$
None	Min-Block Streaming (Alg. 6)	$1/m$	$\mathcal{O}(n)$
Cardinality	Min-Block Greedy (Alg. 8)	$\frac{e-1}{(e-1)m+1}$	$\mathcal{O}(n^2)$
Cardinality	Hierarchical Greedy (Alg. 10)	$(\frac{\tau-1}{2\tau-1})^r \frac{k_r}{\lvert V\rvert}$	Depends on hierarchy.
Cardinality	Card. Round-Robin Greedy (Alg. 11)	$\frac{(1-1/e)^2}{3}$	Depends on a subproblem.
Matroid	Min Block Greedy (Alg. 8)	$\frac{1}{m+2}$	$O(n^2)$
Matroid	Matroid Round-Robin Greedy (Alg. 12)	$\frac{1-1/e}{5}$	$\mathcal{O}(n^2(\log\log m + \log\frac{1}{\delta}))$
p Matroids	Min Block Greedy (Alg. 8)	$\frac{1}{m+p+1}$	$\mathcal{O}(n^2)$
Knapsack	Min Block Greedy (Alg. 8)	$\frac{1}{m+\frac{2}{1-1/e}}$	$\mathcal{O}(n^2)$

The various constraints (e.g., cardinality, matroid, knapsack) we introduce to the robust partition problem greatly improve the applicability of robust submodular partitioning. Several applications that benefit from the constraints include: (1) Partition a training data for machine learning models in distributed training or forming deterministic mini-batches. The additional constraint can be the number of samples from each class to be no more than a certain value. If there are enough samples in the training data, every resulting block will have the same number of samples for each class, which avoids imbalance, further promotes each block's diversity, and improves the consistency of the gradients. (2) Given an undirected graph, we partition the edges into subgraphs so that each subgraph is representative based on the submodular evaluation, and we also constrain each subgraph to have no cycles (a cycle

matroid). A practical scenario is that we wish to send information efficiently over a graph of devices. We partition the graph so that information can be sent in parallel, and the constraint to have no cycles enforces that information is not redundantly sent twice to the same device, leading to improved communications efficiency. (3) Again, for an undirected and connected graph, we partition the edges into subgraphs and constrain that by removing any block in the partition from the original graph, the remaining graph is still connected (a bond matroid). In practice, this works as a form of reliability insurance. For a graph of devices, we partition the graph to perform computation in parallel, and if the connections in one partition fail, the other blocks can still operate/communicate since the graph remains connected.

Related Work

Golovin [39] introduces robust submodular partitioning, which is also often called submodular fair allocation of indivisible goods, and proposes a matching-based algorithm with a bound of $\frac{1}{n-m+1}$. Khot and Ponnuswami [61] proposes a binary search based algorithm and gives an improved bound of $\frac{1}{2m-1}$. Asadpour and Saberi [2] uses an ellipsoid approximation approach and gives a bound of $\Omega(\frac{1}{\sqrt{n}m^{1/4}\log n \log^{3/2} m})$. A Round-Robin Greedy approach is given in [9] with a bound of $\frac{1-e^{-1}}{3}$. Ghodsi et al. [38] proposes a local search algorithm with a bound of $\frac{1}{3}$. Both [9] and [38] requires guessing of the optimal solution value from an exponentially decreasing sequence of values, so strictly speaking, they lose an extra $(1+\delta)$ factor in the approximation bound where $(1+\delta)$ is the exponential factor for the guessing sequence. We can set the δ value small to get close to the constant bounds shown above at the costs of computation. We adapt the Round-Robin Greedy approach [9] to the constrained case. Cotter et al. [24] studies (as well as allowing multiple blocks to be jointly scored) a matroid constrained "groupings" (e.g., coverings, packings) problem but only a fractional subset of groups (rather than the minimum of the groups), is guaranteed to have values larger than the bounded max-min OPT, while our bound compares the min block evaluation to the optimal max-min value.

Another line of related research is the submodular load balancing problem, which mini-

mizes the maximum-valued block in the partition according to the submodular evaluations. In contrast to promoting diversity of each block for the robust submodular partition problem, submodular load balancing enforces every block to contain redundant items, which can be treated as a clustering objective. Theoretically, this problem has been shown to be much harder as Svitkina and Fleischer [123] shows a information theoretical lower bound of $o(\sqrt{\frac{n}{\log n}})$, and also gives a sampling algorithm to match the lower bound up to constant factors. Similar to the max-min case, Ghodsi et al. [38] uses the ellipsoidal approximation to get a bound of $\mathcal{O}(\sqrt{n}\log n)$. Wei et al. [141] gives a Lovász extension based relaxation algorithm and achieves a bound of m.

3.1 Preliminaries and Formulation

With a ground set V of n items, a submodular function f is a set function $2^V \rightarrow \mathbb{R}$ that satisfies the property: $f(A)+f(B) \geq f(A \cup B)+f(A \cap B)$, where $A, B \subseteq V$. Equivalently, a submodular function is characterized by diminishing returns: $f(v|A) \geq f(v|B)\ \forall v \notin B$ and $A \subset B \subseteq V$, where $f(v|B) = f(\{v\} \cup B) - f(B)$. Submodular functions naturally describe the diversity or representativeness of a given set of items. Many simple greedy-based algorithms have been developed to solve optimization problems involving submodular functions, giving both theoretical approximation guarantees, as well as good empirical performance. We restrict the submodular functions discussed in this work to be monotone non-decreasing and normalized, i.e., $f(B) \geq f(A)\ \forall A \subseteq B \subseteq V$, $f(\emptyset) = 0$.

A matroid $\mathcal{M} = (V, \mathcal{I})$ is a set system that describes the independence relationships among the subsets of the ground set V. $\mathcal{I}$ is a set of subsets of V and every $S \in \mathcal{I}$ is considered an independent subset. The matroid rank function is defined as $r_{\mathcal{M}}(A) = \max\{|S| : S \subseteq A, S \in \mathcal{I}\}$. $r_{\mathcal{M}}(V)$ indicates the maximum size of a subset that may be independent according to the matroid $\mathcal{M}$. All subsets of cardinality $\leq k$ with some integer $k > 0$ form a *uniform matroid*, which we denote by $\mathcal{M}_k^u$. A partition matroid is one where we partition the ground set into blocks, and a set is independent if it intersects each block by no more than a block-specific limit. We define a particularly useful partition matroid on

an expanded ground set $\bar{V}$ as follows: We first duplicate the ground set m times, creating $V_1 = V_2 = \ldots V_m = V$, which are ground set copies. We create an expanded ground set $\bar{V} = \uplus_{j=1:m} V_j$ as the disjoint union. A subset $S \subseteq \bar{V}$ is independent in $\mathcal{M}_m^p$ if for every element $v \in V$, let its m copies in $\bar{V}$ be $\{v_1, v_2, \ldots, v_m\}$, we have $|S \cap \{v_1, v_2, \ldots, v_m\}| \leq 1$, i.e., S contains at most one copy of element v. Apart from the uniform matroid and this particular partition matroid, there are many other matroids reflecting a natural notion of independence, for example, the linearly-independent set of real vectors and the spanning trees in a graph. In the below, we use both $S \in \mathcal{M}$ and, when clear, $S \in \mathcal{I}$, to indicate that S is independent in the matroid $\mathcal{M} = (V, \mathcal{I})$.

Matroids are often used as constraints in submodular optimization problems: $\max_{S \in \mathcal{I}} f(S)$ with a matroid $\mathcal{M} = (V, \mathcal{I})$. When $\mathcal{M}$ is a uniform matroid $\mathcal{M}_k^u$, this reduces to the cardinality submodular max and the greedy algorithm gives a $1 - e^{-1}$ bound [29]. When $\mathcal{M}$ is a partition matroid $\mathcal{M}_m^p$, this problem is referred to as the submodular welfare problem (particularly it's the homogeneous case as we assume the same function for each block). For a general constraint with the intersection of p matroids, the greedy algorithm gives a $\frac{1}{p+1}$ bound [33].

Suppose we represent a set S as a binary indicator vector $x_S \in \{0, 1\}^n$, i.e., $\forall i \in [n], x_S[i] = 1$ if $v_i \in S$ or otherwise $x_S[i] = 0$. Then for all the independent sets of a matroid $\mathcal{M} = (V, \mathcal{I})$, the convex hull over all the $x_S, S \in \mathcal{I}$ forms a polytope, which is called the matroid polytope $\mathcal{P}_\mathcal{M}$ of matroid $\mathcal{M}$ [29]. Based on the convex property of the matroid polytope, algorithms [16, 18, 19, 130] have been proposed to firstly solve a continuous extension of the submodular optimization problem under the matroid polytope constraint, which generates a fractional solution in $[0, 1]^n$, and then round the fractional solution to an integral solution to get the resulting set. The Continuous Greedy Algorithm [16] gives a $1 - e^{-1}$ guarantee under a single matroid constraint using the pipage rounding [1, 16]. Interestingly, running the Continuous Greedy under a partition matroid constraint (submodular welfare problem) gives a uniform fractional solution, i.e., on the expanded ground set $\bar{V}$, the fractional solution $x = (\frac{1}{m}, \frac{1}{m}, \ldots, \frac{1}{m})$ (i.e., assigning $\frac{1}{m}$ of every element to each block)

leads to a $1 - e^{-1}$ bound in expectation for an integral solution that assigns each element in V uniformly to one of the m blocks. Such observation also constitutes the basic idea of Round-Robin Greedy for solving the robust submodular partition problem [9], which we will discuss in more detail later.

For a submodular function f on a ground set V, the robust submodular partition problem (submodular fair allocation) [39] is defined as:

$$\max_{\pi \in \Pi(V,m)} \min_{A \in \pi} f(A), \tag{3.1}$$

Where m is the number of blocks in a partition, we denote all possible partitions with m blocks of ground set V as $\Pi(V, m)$, and one partition π with $|\pi| = m$ is a collection of m disjoint sets. Equivalently, we can represent the partition using a partition matroid constraint on the expanded ground set $\bar{V}$:

$$\max_{S \subseteq \bar{V}, S \in \mathcal{M}_m^p} \min_{j \in [m]} f(S \cap V_j). \tag{3.2}$$

Intuitively, the above optimization for robust submodular partition encourages the minimum-valued block to have a high submodular evaluation. Compared to the submodular welfare problem, the robust submodular partition promotes fairness for every one of the partition blocks.

There have been two recent approximation algorithms developed to solve Eq. (3.1). Barman and Krishna Murthy [9] proposes a Round-Robin Greedy algorithm, which iteratively traverses all the blocks in a fixed order, and greedily adds an element with the largest gain to each block. Ghodsi et al. [38] applies a local search approach, which starts with an arbitrary partition and keeps moving an element from a non-minimum block to the minimum block if this relocation improves the objective by certain threshold until no such element can be found. They both require guessing the optimal solution's value, and they need to run multiple instances of their algorithms with the guessed optimal values as an exponentially decreasing sequence from the maximal possible value $f(V)$ to the optimal solution value $\mu = \max_{\pi \in \Pi(V,m)} \min_{S \in \pi} f(S)$. With the exponential decreasing factor as $1 + \delta$, the running

time (in terms of submodular function calls) is $O(n^2 \log \frac{f(V)}{\delta\mu})$ for [9], and $O(n^2m^2 \log \frac{f(V)}{\delta\mu})$ for [38]. We give more details on the running time and optimal value guessing in Sec. 3.3. Min-Block Greedy, a much simpler algorithm, has a running time of $O(n^2)$. Note that the settings of [9] and [38] are slightly more general than Eq. (3.1) as the submodular function for each block can be different. But it's not the heterogeneous case either as they focus on a different notion of optimality.

3.2 Min-Block Greedy Based Algorithms

We propose a Min-Block Greedy Algorithm 7 for Eq. (3.1), which loops over n iterations, and at every iteration, for the minimum-valued block $A_{j^*} \in \text{argmin}_j f(A_j)$, it finds the element with the largest gain $f(v|A_{j^*})$. We prove a $1/m$ bound of Min-Block Greedy. In fact, the proof works for a simpler algorithm, Min-Block Streaming Algorithm 6, which assumes that the algorithm accesses elements from the ground set in an arbitrary order as a stream $V = (v_1, v_2, \ldots, v_n)$, and it assigns the incoming element to the block with the least evaluation. We denote the optimal partition to Eq. (3.1) as $\pi^* = \{O_1, O_2, \ldots, O_m\}$.

Algorithm 6: Min-Block Streaming

input : f, $V = (v_1, v_2, \ldots, v_n)$ as a stream, m

$R := V$;

Let $A_1 = A_2 = \ldots = A_m = \emptyset$;

for $i = 1 : n$ **do**

  $j^* \in \text{argmin}_j f(A_j)$;

  $A_{j^*} := A_{j^*} \cup \{v_i\}$;

return $(A_1, A_2, \ldots, A_m)$

Lemma 1 (Unconstrained Min-Block Streaming). *For a ground set V and its elements $(v_1, v_2, \ldots, v_n)$ coming in an arbitrary streaming order, the output solution of Alg. 6 has $\min_{j\in[m]} f(A_j) \geq \frac{1}{m} \min_{j\in[m]} f(O_j)$.*

Algorithm 7: Min-Block Greedy

input : f, V, m

$R := V$;

Let $A_1 = A_2 = \ldots = A_m = \emptyset$;

while $R \neq \emptyset$ **do**

  $j^* \in \text{argmin}_j f(A_j)$;

  $v^* \in \text{argmax}_{v \in R} f(v|A_{j^*})$;

  $A_{j^*} := A_{j^*} \cup \{v^*\}$;

  $R := R \setminus \{v^*\}$;

return $(A_1, A_2, \ldots, A_m)$

Proof. To prove the guarantee for Alg. 6, we consider the resulting partitioning: $\pi = (A_1^\pi \cup A_2^\pi, \ldots, A_m^\pi)$. For simplicity of notation, we write A_j^π as A_j for each j in the remaining proof. We refer OPT to the optimal solution for this proof, i.e., $OPT = \max_\pi \min_j f(A_j)$. W.l.o.g., we assume $f(A_1) = \min_i f(A_j)$. Let a_j be the last item to be chosen in block A_j for $j = 2, \ldots, m$.

Claim 1:

$$OPT \leq f(V \setminus \{a_2, \ldots, a_m\}) \tag{3.3}$$

To show this claim, consider the following: If we enlarge the singleton value of $a_j, j = 2, \ldots, m$, we obtain a new submodular function:

$$f'(A) = f(A) + \alpha \sum_{j=2}^{m} |A \cap a_j|, \tag{3.4}$$

where α is sufficiently large. Then running StreamGreed on f' with the same ordering of the incoming items leads to the same solution, since only the gain of the last added item for each block is changed.

Note that $f'(A) \geq f(A), \forall A \subseteq V$, we then have $\max_\pi \min_j f'(A_j^\pi) \geq OPT$. The optimal partitioning for f' can be easily obtained as $\pi' = (V \setminus \{a_2, \ldots, a_m\}, a_2, \ldots, a_m)$. Therefore,

we have that

$$OPT \leq \max_{\pi} \min_{j} f'(A_j^{\pi}) \tag{3.5}$$

$$= f'(V \setminus \{a_2, \ldots, a_m\}) = f(V \setminus \{a_2, \ldots, a_m\}). \tag{3.6}$$

Lastly, we have that $f(A_1) \geq f(A_j \setminus a_j)$ for any $j = 2, \ldots, m$ due to the procedure of StreamGreed. Therefore we have the following:

$$f(A_1) \geq \frac{1}{m}(f(A_1) + \sum_{j=2}^{m} f(A_j \setminus a_j)) \tag{3.7}$$

$$\geq \frac{1}{m} f(V \setminus \{a_2, \ldots, a_m\}) \text{ // submodularity of } f \tag{3.8}$$

$$\geq \frac{1}{m} OPT \text{ // Claim 1} \tag{3.9}$$

□

Corollary 1 (**Unconstrained Min-Block Greedy).** *The output solution of Alg. 7 has* $\min_{j \in [m]} f(A_j) \geq \frac{1}{m} \min_{j \in [m]} f(O_j)$ *since the order of adding elements in Min-Block Greedy is one possible order of the ground set elements.*

Intuitively, Alg. 7 optimizes the objective Eq. (3.1) greedily, i.e., it always increases the current value (the minimum-block evaluation) with the largest possible gain, while the performance of Alg. 6 greatly depends on the order of elements, so it might seem that the bound for Min-Block Greedy should improve upon the current $\frac{1}{m}$ bound. However, as shown in our result in the following lemma, the bound in Corollary 1 is tight.

Lemma 2 (**Tightness of Corollary 1).** $\forall \epsilon > 0$, $\exists$ *a submodular function* f *such that the output solution of Alg 7* $\min_{j=1:m} f(A_j) = \frac{1}{m} \min_{j=1:m} f(O_j) + \epsilon$.

We elaborate on how to construct the submodular function below. The key idea is that we can find a set-cover function where even though Min-Block Greedy selects the element with the largest gain, the element can still be quite redundant with the current minimum

block. Say the current minimum block is A, and the maximum-gain element is v, because of the greedy step, v is larger in terms of $f(v|A)$ than other elements $v' \in R \setminus v$, but $\frac{f(v|A)}{f(v)}$ can be very small, i.e., the area covered by v according to the set-cover function is already mostly covered by A. On the other hand, the optimal solution can fully utilize $f(v)$ and makes v cover a much larger area overall. Note that Lemma 2 also serves as the tightness for Lemma. 1 since the order of adding elements in Min-Block Greedy follows a streaming order.

Proof. We construct a set cover function as the tight example for Corollary 1. We illustrate the set cover function graphically in Fig. 3.1.

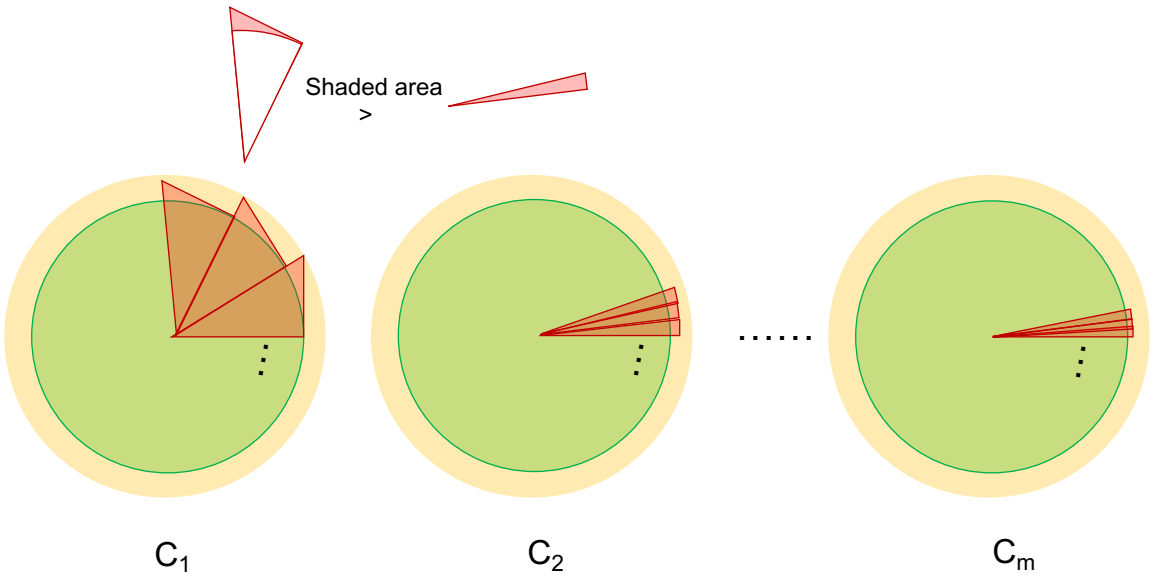


Figure 3.1: A graphical illustration of the tight example. The circles are the areas to cover for the set cover function and the green inner circles and the red triangles are elements in the ground set (the outer yellow circles are not elements). The inner circles (green) largely overlap with the outer circles (yellow). The red triangles mostly overlap with the inner circle, with little gains on the ring between the two circles. We can change the size of the red triangles so that Min-Block Greedy prefers a redundant element (the shaded area comparison on the top of the figure). Also note that the red triangles may overlap on the inner circle part (they may not retain the shapes as triangles), so overall they cover $m-1$ times the area of each circle.

Suppose we have m circles of area to cover in the set cover function. Say we order the circles by their area, say $C_1 < C_2 < \ldots < C_m$. Let $C_{j+1} = C_j + \epsilon_{j+1}$ for some $\epsilon_{j+1} > 0$. For every circle j, we have an element $v_j \in V$, which covers an inner circle, which almost covers the entire circle. W.l.o.g, suppose $f(v_1) = 1$ and let $C_j = f(v_j) + \delta_j$.

For each circle C_j, we construct n_j elements, which largely overlap with the inner circle covered by v_j and gives little gain on the ring between the inner circle and the outer circle C_j. Call these n_j elements V_j. Let $f(V_j|v_j) = \epsilon'_j$, $f(v) < f(v_j) \forall v \in V_j$, and $f(V_j) > f(v_j)$.

Now let's focus on the first two circles C_1 and C_2, and assume $m = 2$ for the partition problem. It is easy to extend to general m case by recursively applying the following arguments on C_2 and C_3.

Suppose we run the min-block greedy, after the first two steps, one block contains v_1 and the other contains v_2. At step 3, v_1 is the min-block. By setting the suitable values for n_1 and n_2 (say $n_2 >> n_1$), we can make $f(v|v_1) > f(v'|v_1) \forall v \in V_1, v' \in V_2$. Therefore, we will still select an element from V_1 even though such element overlaps largely with the inner circle v_1. We can force the min-block greedy algorithm to select all elements from V_1 before the min-block changes to the block containing v_2, and $f(V_1 \cup v_1) > f(v_2)$. After that, the algorithm can only add elements from V_2 to the block containing v_2, which gives only ϵ'_2 gains. As we can make the values of ϵ_j, ϵ'_j and δ_j arbitrarily small, the solution is arbitrarily close to 1. On the contrary, the optimal partition should add elements in V_1 to v_2 and elements in V_2 to v_1, and the solution has value arbitrarily close to 2.

To extend to general m partitions, we may treat the current V_2 as V_1, and construct V_3 in the same way we construct V_2 based on V_1. In the first m steps of the min-block greedy, the algorithm is forced to evenly distribute $v_j, j = 1, 2, \ldots, m$ into every block. After that, the algorithm adds all elements in V_j to the block containing v_j before the min-block changes and the block containing v_j will not become the min-block again. In the end, every block only covers (almost) one circle. Suppose for V_j, we may make the elements to cover the inner circle multiple times, i.e., $\exists \pi \in \Pi(V_j, m) s.t. \forall A \in \pi, f(A) \geq f(v_j)$. Then for the optimal solution, every block can cover (almost) all the circles, and therefore the approximation ratio can be

arbitrarily close to $1/m$.

□

More generally, given a constraint $\mathcal{C}$, we define the constrained robust submodular partition as:

$$\max_{\pi \in \Pi(V,m,\mathcal{C})} \min_{A \in \pi} f(A), \tag{3.10}$$

Where $\Pi(V, m, \mathcal{C})$ is the set of all possible partitions on set V into m blocks such that for every partition $\pi \in \Pi(V, m, \mathcal{C})$, every block $A \in \pi$ should satisfy the constraint $A \in \mathcal{C}$. We denote the optimal partition to Eq. (3.10) as $\pi_C^* = \{O_1^{\mathcal{C}}, O_2^{\mathcal{C}}, \ldots, O_m^{\mathcal{C}}\}$. We remark that due to the constraint, not all elements can get assigned to some block in the solution, so strictly speaking the solution is an allocation of elements rather than a partition.

For now, we will take $\mathcal{C}$ as any down-closed constraint: Let $\mathcal{C}$ be a collection of subsets of the ground set V, and by satisfying the constraint, we require the solution A to be one of the subsets in $\mathcal{C}$. The down-closed property means that if $A \in \mathcal{C}$ we have $B \in \mathcal{C}$ for any $B \subseteq A$. Following Eq. (3.10), we can define the constrained problem in terms of the expanded subset $\bar{V}$:

$$\max_{S \subseteq \bar{V}, S \in \mathcal{M}_m^p, \forall j: (S \cap V_j) \in \mathcal{C}} \min_{j \in [m]} f(S \cap V_j). \tag{3.11}$$

Based on the Min-Block Greedy algorithm for the unconstrained case, we propose a natural extension to the constrained case (Alg. 8), where at every iteration, for the minimum-valued block A_{j^*}, we greedily find the element v^* that still keeps the block feasible under the constraint $\mathcal{C}$, i.e., $\{v^*\} \cup A_{j^*} \in \mathcal{C}$. If we cannot find any element in the remaining set to add to the current min-block, we remove the current block from the candidate blocks and move to the next smallest-valued block. In Line 9-11 of Alg. 8, we have an extra step of comparing the solution to the largest singleton element in the remaining elements.

In Line 7 of Alg. 8, we call a subroutine GreedyStep$(R, \mathcal{C}, A_{j^*})$ to greedily find a feasible element. The subroutine varies according to the type of constraint $\mathcal{C}$. Particularly, for the

Algorithm 8: Constrained Min-Block Greedy

input : submodular function f, ground set V, number of blocks m, constraint $\mathcal{C}$

$R := V$;
Let $A_1 = A_2 = ... = A_m = \emptyset$;
Let $J = [m]$;
while $R \neq \emptyset$ *and* $J \neq \emptyset$ **do**
  $j^* \in \text{argmin}_{j \in J} f(A_j)$;
  if $\exists v \in R$ *s.t.* $A_{j^*} \cup \{v\} \in \mathcal{C}$ **then**
    $v^* := \text{GreedyStep}(R, \mathcal{C}, A_{j^*})$;
  else
    $a^* \in \text{argmax}_{a \in A_{j^*} \cup R} f(\{a\})$;
    if $f(\{a^*\}) \geq f(A_{j^*})$ **then**
      $A_{j^*} := \{a^*\}, R := R \setminus \{a^*\}$;
    Let $J = J \setminus j^*$;
    Continue;
  $A_{j^*} := A_{j^*} \cup \{v^*\}$;
  $R := R \setminus \{v^*\}$;
return $(A_1, A_2, ..., A_m)$

constrained submodular maximization problem defined as

$$\max_{S \subseteq V, S \in \mathcal{C}} f(S), \tag{3.12}$$

GreedyStep$(\cdot)$ is shared by Alg. 8 and Alg. 9, and if Alg. 9 is an approximation algorithm of solving Eq. (3.12) with some bound α, we can prove the following result for Alg. 8.

Theorem 8 (**Constrained Min-block Greedy**). *Given a constraint $\mathcal{C}$, if the greedy solution S^g to problem $\max_{S \in \mathcal{C}} f(S)$ using Alg. 9 has a bound of α, i.e., $f(S^g) \geq \alpha \max_{S \in \mathcal{C}} f(S)$, then the solution of Alg. 8 has $\min_{j \in [m]} f(A_j) \geq \frac{\alpha}{\alpha m + 1} \min_{j \in [m]} f(O_j^{\mathcal{C}})$.*

Algorithm 9: Constrained Submodular Greedy Max

input : submodular function f, ground set V, constraint $\mathcal{C}$

1 $R := V$;
2 Let $S^g = \emptyset$;
3 **while** $R \neq \emptyset$ **do**
4 **if** $\exists v \in R$ *s.t.* $A_{j^*} \cup \{v\} \in \mathcal{C}$ **then**
5 $v^* := \text{GreedyStep}(R, \mathcal{C}, S^g)$;
6 **else**
7 Break;
8 $S^g := S^g \cup \{v^*\}$;
9 $R := R \setminus \{v^*\}$;
10 $a^* \in \text{argmax}_{a \in V} f(\{a\})$;
11 **return** $\text{argmax}_{A \in \{S^g, \{a^*\}\}} f(A)$

Proof. W.l.o.g., we assume that the block of index 1 for a partition corresponds to the minimum-valued block, e.g., $f(O_1^{\mathcal{C}}) = OPT^{\mathcal{C}}$. For Min-Block Greedy algorithm, we always add an element feasible to the constraint $\mathcal{C}$ to the block with the minimum evaluation. Let the minimum block in our final solution be A_1. Due to the final singleton comparison step (line 9-11 in Alg. 8), there are several different scenarios for A_1:

1. It is never the case that we cannot add any elements to a block due to the constraint (line 6 always true). This is the simplest case as we can directly reduce it to a stream of the elements with the same ordering as we add them into different blocks, and Lemma. 1 applies. We therefore can get an $1/m$ approximation ratio, which is better than the one given in the theorem for any $\alpha \leq 1$.

2. A_1 is the first block that we cannot find any feasible elements to add. The singleton comparison step may increase the function value of A_1. however, by assumption it's

still the minimum block after the algorithm completes.

3. There are other blocks that we cannot find any feasible elements to add before A_1. This could only happen if the other blocks get their values increased by the singleton comparison step. As if the singleton comparison step does not swap the block with the largest singleton, the block, which is not A_1 in this case, is the minimum block for that step and remains minimum for the following steps of the algorithm.

For scenarios 2 and 3, the general idea of the proof is the same, where we separate the ground set V into two parts V' and R' ($V = V' \cup R'$), and bound $f(A_1)$ by comparing to a block in the optimal solution $O_j^{\mathcal{C}}$ through $f(O_j^{\mathcal{C}} \cap V')$ and $f(O_j^{\mathcal{C}} \cap R')$. However, for 2 and 3, we will use slightly different V' and R'.

First, for scenario 2), let's suppose at step t', the current minimum block is A_1, and we find no feasible elements to add. Let all the elements allocated so far (before the singleton comparison step for A_1) as V', and the remaining unallocated elements as R'. $V = V' \cup R'$. Denote the elements in A_1 before the singleton comparison step as A_1', as the singleton step always improves the block value, we have $f(A_1) \geq f(A_1')$.

If we run the min-block robust partition greedy algorithm on V' only, we will get the same partial partition as we run on V for t' steps. Therefore, suppose we create a stream that orders the elements in V' in the same order that those elements get allocated by the min-block robust partition greedy algorithm, then by Lemma. 1, we have:

$$f(A_1) \geq f(A_1') \geq \frac{1}{m} OPT(V'), \tag{3.13}$$

Where we denote $OPT(V') = \max_{\pi \in \Pi(V', m)} \min_{A \in \pi} f(A)$ as the optimal solution for the unconstrained robust submodular partition on the ground set V'.

Let $O_j^{\mathcal{C}}$ be some block in the optimal constrained partition on ground set V. since $O_j^{\mathcal{C}}$ can be the non-minimal block in the optimal solution, we have:

$$f(O_j^{\mathcal{C}}) \geq OPT^{\mathcal{C}}. \tag{3.14}$$

There exists a $j \in \{1, \ldots, m\}$ such that

$$f(A_1) \geq \frac{1}{m} OPT(V') \tag{3.15}$$

$$\geq \frac{1}{m} f(O_j^{\mathcal{C}} \cap V'), \tag{3.16}$$

as otherwise $\forall j \in \{1, \ldots, m\}, O_j^{\mathcal{C}} \cap V'$ forms a solution for the partition problem on the reduced ground set V', and gives a solution value better than $OPT(V')$, which violates the optimality of $OPT(V')$.

Now we separate the constrained optimal solution on ground set V into 2 parts: $O_j^{\mathcal{C}} \cap V'$ and $O_j^{\mathcal{C}} \cap R'$.

Assumption 1. Suppose

$$f(O_j^{\mathcal{C}} \cap R') \geq f(O_j^{\mathcal{C}} \cap V'), \tag{3.17}$$

Then because of submodularity, $f(O_j^{\mathcal{C}} \cap R') + f(O_j^{\mathcal{C}} \cap V') \geq f(O_j^{\mathcal{C}})$ (recall $V = V' \cup R'$)and we have

$$f(O_j^{\mathcal{C}} \cap R') \geq \frac{1}{2} f(O_j^{\mathcal{C}}). \tag{3.18}$$

Consider the set $R' \cup A_1'$, let

$$\hat{O} \in \underset{S \subseteq R' \cup A_1', S \in \mathcal{C}}{\operatorname{argmax}} f(S), \tag{3.19}$$

I.e., $\hat{O}$ is the optimal solution to the constraint submodular max on the reduced ground set $R' \cup A_1'$. After the singleton comparison step on A_1', we get A_1, which is the greedy solution of Alg. 8 on the reduced ground set $R' \cup A_1'$ and constraint $\mathcal{C}$. Therefore, based on the α-bound assumption in Theorem 8, we have:

$$f(A_1) \geq \alpha f(\hat{O}) \tag{3.20}$$

$$\geq \alpha f(O_j^{\mathcal{C}} \cap R') \tag{3.21}$$

$$\geq \frac{\alpha}{2} f(O_j^{\mathcal{C}}) \tag{3.22}$$

$$\geq \frac{\alpha}{2} f(O_1^{\mathcal{C}}). \tag{3.23}$$

Eq. (3.21) comes from the optimality of $\hat{O}$ and Eq. (3.22) comes from **Assumption 1**.

Assumption 2. Otherwise, we have

$$f(O_j^{\mathcal{C}} \cap V') > f(O_j^{\mathcal{C}} \cap R') \tag{3.24}$$

$$\geq \frac{1}{2} f(O_j^{\mathcal{C}}). \tag{3.25}$$

We therefore have:

$$f(A_1) \geq \frac{1}{m} f(O_j^{\mathcal{C}} \cap V') \tag{3.26}$$

$$> \frac{1}{2m} f(O_j^{\mathcal{C}}) \tag{3.27}$$

$$\geq \frac{1}{2m} f(O_1^{\mathcal{C}}) \tag{3.28}$$

Note that one of **Assumption 1** and **Assumption 2** is always true, since $f(O_j^{\mathcal{C}} \cap R') + f(O_j^{\mathcal{C}} \cap V') \geq f(O_j^{\mathcal{C}})$ because of submodularity. Previously, we use equal weights of $\frac{1}{2}$ for both assumptions. We can balance the weights as long as the weights sum to one, and we get:

if $f(O_j^{\mathcal{C}} \cap R') \geq \frac{1}{\alpha m+1} f(O_j^{\mathcal{C}})$, we have

$$f(A_1) \geq \alpha f(\hat{O}) \tag{3.29}$$

$$\geq \alpha f(O_j^{\mathcal{C}} \cap R') \tag{3.30}$$

$$\geq \frac{\alpha}{\alpha m + 1} f(O_j^{\mathcal{C}}) \tag{3.31}$$

$$\geq \frac{\alpha}{\alpha m + 1} f(O_1^{\mathcal{C}}); \tag{3.32}$$

if $f(O_j^{\mathcal{C}} \cap V') > \frac{\alpha m}{\alpha m+1} f(O_j^{\mathcal{C}})$, we have

$$f(A_1) > \frac{1}{m} f(O_j^{\mathcal{C}} \cap V') \tag{3.33}$$

$$> \frac{1}{m} \frac{\alpha m}{\alpha m + 1} f(O_j^{\mathcal{C}} \cap V) \tag{3.34}$$

$$> \frac{\alpha}{\alpha m + 1} f(O_1^{\mathcal{C}}). \tag{3.35}$$

Thus, we get a $\frac{\alpha}{\alpha m+1}$ bound.

For scenario 3, we only need to change V' and R' and the same argument follows. Recall that in such a scenario, there are some other blocks that have no feasible elements to add before A_1, and they get their values increased through the singleton comparison step. There are also two different cases here. Firstly, it does not happen that there are no feasible elements to add to A_1 until the end of the algorithm. In such a case, similar to the scenario 1, we can order the elements as a stream and applies Lemma. 1 to get the $1/m$ approximation ratio. Note that the blocks that get to the singleton comparison step all get their values increased for scenario 3, and we can just add the singleton a^* (line 9) to that block in the streaming case. To be more precise, for Alg. 8 when block j ($j \neq 1$) is the current minimum block, and has no feasible elements to add, we denote its elements as A'_j, and the singleton comparison step gives an element a^* with $f(\{a^*\}) \geq f(A'_j)$. In the streaming ordering, we use the same ordering as we add element sin Alg. 8, and at the singleton comparison step for block j, we have the next element in the stream be a^*, and we add that element to block j since block j is the current minimum block. By monotonicity, we have $f(A'_j \cup \{a^*\}) \geq f(\{a^*\}) \geq f(A_1)$. In other words, those blocks never become the minimum block again, and no elements get added to them after their singleton comparison step. Therefore, we have a streaming ordering of the elements that will make the minimum block equal to A_1 and Lemma. 1 applies.

Next, we discuss for the case where it happens that there are no feasible elements to add to A_1. When that happens, we set all the allocated elements as V' and the remaining elements as R' before the singleton comparison step. Note for those blocks that get to the singleton comparison step before A_1, we will also include the singletons in V' (recall those singletons have larger gains and get swapped with the elements in those blocks for this scenario). For such V' and R', the exact argument in scenario 2 can be made, i.e., we can treat A'_1 as a min-block streaming solution on V' (Eq. 3.26) and A_1 as a greedy solution on $A'_1 \cup R'$ (Eq. 3.21).

□

Corollary 2 (**Cardinality Constrained Min-block Greedy**). *For $\mathcal{C}$ as a cardinality constraint, the output of Alg 8 has* $\min_{j=1:m} f(A_j) \geq \frac{1}{m+\frac{1}{1-e^{-1}}} \min_{j=1:m} f(O_j^{\mathcal{C}})$.

Corollary 3 (**Matroid Constrained Min-block Greedy**). *For $\mathcal{C}$ as an intersection of p matroids constraint, the output of Alg 8 has* $\min_{j=1:m} f(A_j) \geq \frac{1}{m+p+1} \min_{j=1:m} f(O_j^{\mathcal{C}})$.

Corollary 4 (**Knapsack Constrained Min-block Greedy**). *For $\mathcal{C}$ as a knapsack constraint, the output of Alg 8 has* $\min_{j=1:m} f(A_j) \geq \frac{1}{m+\frac{2}{1-1/e}} \min_{j=1:m} f(O_j^{\mathcal{C}})$.

For $\mathcal{C}$ as a cardinality constraint, GreedyStep$(\cdot)$ just picks the element with the largest gain until the block reaches the cardinality limit k. For $\mathcal{C}$ as an intersection of p matroid constraints, GreedyStep$(\cdot)$ finds the element v^* that has the largest gain $f(v^*|A_{j^*})$ while keeping the block still feasible, i.e., $v^* \cup A_{j^*} \in \mathcal{C}$. For $\mathcal{C}$ as a knapsack constraint with the weight of each element v as $w(v)$, GreedyStep$(\cdot)$ finds the element v^* with the largest ratio $\frac{f(v^*|A_{j^*})}{w(v)}$ while keeping the sum of weights below the given budget. Due to the tightness of the $\frac{1}{m}$ bound we have proved for the unconstrained case, the $\frac{1}{m}$ dependence in the constrained bound cannot be improved.

For $\mathcal{C}$ as a cardinality constraint, we prove an improvement bound for the Min-block greedy algorithm compared to Corollary 2 using Algorithm 8.

Theorem 9 (**Min-Block Cardinality Constraint**). *For submodular function f on ground set V and block size (mini-batch size) constraint k, suppose $m = |V|/k$, Algorithm 8 gives an approximation ratio of $\frac{e-1}{(e-1)m+1}$.*

Proof. At each iteration of Algorithm 8, it finds the smallest partially filled block (i.e., block size is less than k) and assigns the element with the largest gain to that block. The objective value we care about is the minimum block value. Therefore, at the end of some iteration (line 15 of Algorithm 8), if we find j s.t. $|A_j| = k$ where $j \in \text{argmin}_j f(A_j)$, we know the final minimum block value is $f(A_j)$, even though there are still elements unassigned at that moment (i.e. $|R| > 0$). The reason is quite simple, block A_j is full so its value will not change

afterwards and all other block values will not decrease since f is monotone non-decreasing. A_j will be minimum ever after. Therefore, in this proof and also for the proof of Theorem 10, we assume that we are at line 15 of the earliest iteration when there exists j s.t. $|A_j| = k$ where $j \in \text{argmin}_j f(A_j)$. We pause the algorithm and prove $\min_j f(A_j)$ has a lower bound at that time. This lower bound will also apply for the final $\{A_j\}s$.

This assumption is useful since at the beginning of current and all previous iterations, we know that for all $j \in \text{argmin}_j f(A_j)$, $|A| < k$. Then $j^* \in \text{argmin}_{j,|A_j|<k} f(A_j)$ is equivalent with $j^* \in \text{argmin}_j f(A_j)$, which will make the analysis simpler. Unfortunately, this is not the end of Algorithm 8, so that $A_1, A_2, \ldots, A_m$ may not partition the entire ground set V, or more precisely, $\cup_{i=1}^m A_i \subseteq V$. We will be mindful of this in the below.

We begin the proof with several definitions and lemmas.

Definition 3. *Let* OPT_g *be* $\max_{\pi \in \Pi(V,k)} \min_i g(\pi_i(V))$ *for submodular function* g *defined on ground set* V*. Let* $OPT = OPT_f$ *for simplicity.*

Definition 4. *Let* $f_A(X) : 2^V \to \mathbb{R}$ *be a set function with* $f_A(X) = f(X \cap A)$*, and* $X \subseteq V$*.*

Immediately, we notice that if f is monotone non-decreasing submodular, so is f_A. Those two definitions are only valid in the scope of this section.

Lemma 3. *For monotone non-decreasing submodular function* f*, for any* $A \subseteq V$*,*

$$OPT_f \leq OPT_{f_{V \setminus A}} + f(A)$$

.

Proof. First, we have $f(X) \leq f(X \setminus A) + f(X \cap A) \leq f_{V \setminus A}(X) + f(A)$ for all $X \subseteq V$ according to submodularity and monotonicity.

Let $\pi^* \in \text{argmax}_{\pi \in \Pi(V,k)} \min_i f(\pi_i(V))$. Then we have $\text{OPT}_{f_{V \setminus A}} \geq \min_i f_{V \setminus A}(\pi_i^*(V)) \geq \min_i [f(\pi_i^*(V)) - f(A)] = \text{OPT}_f - f(A)$.

□

Lemma 4. *For monotone non-decreasing submodular function f,*

$$OPT_f \leq \min_{X \subseteq V, |X|=m-1} \max_{Y \subseteq V \setminus X, |Y| \leq k} f(Y)$$

.

Proof. Let $\pi^* \in \text{argmax}_{\pi \in \Pi(V,k)} \min_i f(\pi_i(V))$. Note that $\text{OPT}_f \leq f(\pi_i^*(V))$ for $i = 1, \ldots, m$.

For all $X \subseteq V$ with $|X| = m-1$, there is at least one $\pi_i^*(V)$ that does not contain any element from X since $\{\pi_i^*(A)\}_{i=1}^m$ are m disjoint sets. Assume $\pi_j^*(V) \cap X = \emptyset$, then $\pi_j^*(V) \subseteq V \setminus X$. Since $|\pi_j^*(V)| \leq k$, we have $\text{OPT}_f \leq f(\pi_j^*(V)) \leq \max_{Y \subseteq V \setminus X, |Y| \leq k} f(Y)$. Note that this holds for all $X \subseteq V$ with $|X| = m-1$. Therefore, $\text{OPT}_f \leq \min_{X \subseteq V, |X|=m-1} \max_{Y \subseteq V \setminus X, |Y| \leq k} f(Y)$

□

Now, we move to the main part of the proof for Theorem 9. We assume $m \in \text{argmin}_{i=1,2,\ldots,m} f(A_i)$ w.l.o.g. (assume the last block has the min evaluation). So $\min_{i=1,2,\ldots,m} f(A_i) = f(A_m)$.

For $i = 1, 2, \ldots, m-1$, assume A_i is not empty; otherwise $|V| < m$ and $\text{OPT}_f = 0$ which immediately proves the theorem. Let a_i be the last added element in block A_i. We claim $f(A_i \setminus \{a_i\}) \leq f(A_m)$, as the greedy process of the algorithm always puts an element into the block with minimal evaluation.

Let $f'(X) = f_{V \setminus [\cup_{i=1:m-1}(A_i \setminus \{a_i\})]}(X)$. We note that

$$\text{OPT}_{f'} \leq \max_{Y \subseteq V \setminus \{a_1, a_2, \ldots, a_{m-1}\}, |Y| \leq k} f'(Y) \tag{3.36}$$

$$= \max_{Y \subseteq V', |Y| \leq k} f(Y) \tag{3.37}$$

according to lemma 4, where $V' = V \setminus (\cup_{i=1:m-1} A_i)$. Next, we want to prove $\text{OPT}_{f'} \leq \frac{1}{\gamma} f(A_m)$.

If $|V'| \leq k$, then $\text{OPT}_{f'} \leq f(A_m)$.

If $|V'| > k$, then $|A_m| = k$. Recall that in the greedy process, at each step, we always choose the block with minimal evaluation and add the element with the largest gain from the remaining set R. Let us focus only on the m-th block. Let $A_m = \{v_1, v_2, \ldots, v_k\}$ where each element is labeled in the greedy order. Assume that just before v_i is picked,

the current remaining set of elements is R_i. Given the greedy process, we have $v_i \in \text{argmax}_{v \in R_i} f(v|\{v_1, v_2, \ldots, v_{i-1}\})$ for all $i = 1, 2, \ldots, k$. We notice that $v_i \in B_i \cap V'$ for all i and $B_i \cap V' = V' \setminus \{v_1, v_2, \ldots, v_{i-1}\}$. Therefore,

$$v_i \in \underset{v \in V' \setminus \{v_1, v_2, \ldots, v_{i-1}\}}{\text{argmax}} f(v|\{v_1, v_2, \ldots, v_{i-1}\}) \tag{3.38}$$

for all $i = 1, 2, \ldots, k$.

Interestingly, this is just the greedy algorithm that chooses k elements from V'. Immediately, we have

$$f(A_m) \geq \gamma \max_{Y \subseteq V', |Y| \leq k} f(Y) \tag{3.39}$$

where $\gamma = 1 - \frac{1}{e}$. Therefore, $\text{OPT}_{f'} \leq \frac{1}{\gamma} f(A_m)$.

Then we use lemma 3,

$$\begin{aligned} \text{OPT}_f &\leq \text{OPT}_{f'} + f(\cup_{i=1,2,\ldots,m-1}(A_i \setminus \{a_i\})) && (3.40)\\ &\leq \text{OPT}_{f'} + \sum_{i=1,2,\ldots,m-1} f(A_i \setminus \{a_i\}) && (3.41)\\ &\leq \frac{1}{\gamma} f(A_m) + (m-1) f(A_m) && (3.42)\\ & && (3.43) \end{aligned}$$

since $f(A_i \setminus \{a_i\}) \leq f(A_m)$.

Therefore, we have

$$\begin{aligned} \min_{i=1,\ldots,m} f(A_i) = f(A_m) &\geq \frac{1}{\frac{1}{\gamma} - 1 + m} \text{OPT}_f && (3.44)\\ &= \frac{e-1}{(e-1)m + 1} \text{OPT}_f && (3.45) \end{aligned}$$

□

Again for $\mathcal{C}$ as a cardinality constraint, we propose a variant of the Min-Block Greedy algorithm that utilizes a hierarchical structure (Alg. 10). By using th hierarchical structure, we can further decrease the computation and memory cost of the original Min-Block Greedy

Algorithm 10: Hierarchical Submodular Robust Partitioning

input : f, V, $k_1, ..., k_r$

1 $k_0 := |V|$; $Q_1 := (V)$; **for** $i := 1;\ i \leq r;\ i := i + 1$ **do**

2 $\quad m_i := k_{i-1}/k_i$; $Q_{i+1} = ()$ **for** $j := 1;\ j \leq |Q_i|;\ j := j + 1$ **do**

3 $\quad\quad A_1, ..., A_{m_i} = \text{MinBlockGreedy}(f, Q_i[j], k_i)$; Append $A_1, ..., A_{m_i}$ to Q_{i+1};

4 $\quad$ **end**

5 **end**

6 **return** Q_{r+1}

algorithm. We will discuss this in more details in Chapter 4, where we use this algorithm on real world tasks with ground set size exceeding one million.

Without any further assumption, we actually show that the approximation ratio can be arbitrarily bad for Algorithm 10 by a simple example in the following proof. However, under mild assumptions about the data points and the function f, i.e., if we assume that in the process of calling Algorithm 8 from Algorithm 10 blocks are not filled in an extremely imbalanced way (which we find not typically occur in practice), then we have the following bound for Algorithm 10.

Definition 5. *We run Algorithm 8 with ground set V', block size constraint k' and $m' = |V'|/k'$, the greedy step (line 5) gets executed $T = |V'|$ times. and we get a sequence of sets $Q = (A_1^T, A_2^T, \ldots, A_{m'}^T)$ as the output, with $A_j'^T$ having the minimal evaluation, i.e. $j' \in \text{argmin}_{i=1:m'} f(A_i^T)$. There exists an earliest greedy step t ($1 \leq t \leq T$), such that $|A_{j'}^t| = k'$, and $j' \in \text{argmin}_{1:m'} f(A_i^t)$ (A_i^t is the ith block at greedy step t), we define $\tau := \min_{i=1:m'} |A_i^t|$.*

Theorem 10 (**Min-Block Hierarchy**). *If we have $\tau \geq 2$ as defined in Def. 5 for every call to Algorithm 8 from Algorithm 10, then we achieve an approximation ratio of $(\frac{\tau-1}{2\tau-1})^r \frac{k_r}{|V|}$.*

Proof. First, we show a simple counter example which can make the bound arbitrarily bad for Algorithm 8 without any partitions. Let $|V| = 8$, $m = 4$, $k_1 = 4$, $k_2 = 2$ and f is modular.

The eight elements have weights $1-2\epsilon, 0.25, 0.25, 0.25, 0.25, \epsilon, \epsilon, \epsilon$ respectively, with $\epsilon > 0$ as a small value. Clearly, OPT=$0.25+\epsilon$, yet Algorithm 10 will give blocks with weights $(0.25, 0.25), (0.25, 0.25), (1-2\epsilon, \epsilon)$ and (ϵ, ϵ), and the min block evaluation is 2ϵ.

Next, we prove Theorem 10 under certain mild assumption.

Definition 6. *For any monotone non-decreasing submodular f and number of partitions M, Let OPT_M be $\max_{\pi \in \Pi(V,|V|/M)} \min_i f(\pi_i(V))$.*

The previous definition is valid only within the scope of this section.

Lemma 5. *For any monotone non-decreasing submodular f, and $M_1, M_2 \in \mathbb{Z}^+$, if $\frac{M_2}{M_1} \in \mathbb{Z}^+$, we have $OPT_{M_1} \geq OPT_{M_2}$.*

Proof. For any optimal partition of M_2, we can group them into M_1 partitions and each partition's function value is greater or equal to OPT_{M_2}. □

In Algorithm 10, there are r iterations and the partition will form a tree structure (Figure 4.2). The root is V and $Q_1 = \{V\}$. In the i-th iteration, it will partition every block A in $Q_i = \{A_{i,j}\}_{j=1}^{|Q_i|}$ into m_i smaller sets and those $|Q_i|m_i$ sets form Q_{i+1}. Immediately, we have that $|Q_i| = \Pi_{j=1}^{i} m_{j-1} = \frac{|V|}{k_{i-1}}$, where m_0 is set to 1. Let $M_i = |Q_i|$ and Q_i is a M_i-partition.

Definition 7. *For any $i = 1, \ldots, r$, $A \in Q_i$ is partitioned into $A_1, A_2, \ldots, A_{m_i}$, we call $child(A) = \{A_1, A_2, \ldots, A_{m_i}\}$ and $parent(A_j) = A$.*

Algorithm 10 is repeatedly calling Algorithm 8, and as we have discussed, we can end Algorithm 8 early to achieve the desired bound. The end condition is that there exists j s.t. $A_j \in \mathrm{argmin}_i A_i$ and $|A_j| = k$. At the time we early stop Algorithm 8, we name the resulting blocks, whose sizes are possibly less than the cardinality constraint, as $\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_{|V|/k}$. We notice $\bar{A}_j \subseteq A_j$ and $\min_i A_i = \min_i \bar{A}_i$. We assume that Algorithm 8 also outputs $\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_{|V|/k}$.

Definition 8. *For any $i = 1, \ldots, r$, $A \in Q_i$, after running Algorithm 8 on A, let $t(A) = \min_{i=1}^{m_i} |\bar{A}_i|$.*

Definition 9. *For $i = 1, \ldots, r$ and $\tau \geq 2$ is an integer, Property $P(i, \tau)$ is true if and only if $t(A) \geq \tau$ for all $A \in Q_i$.*

Definition 10. *For $i = 1, \ldots, r+1$ and $\tau \geq 2$ is an integer, Property $G(i, \tau)$ is true if and only $f(A) \geq \frac{1}{(\frac{2\tau-1}{\tau-1})^{i-1} M_i} OPT_{M_i}$ for all $A \in Q_i$.*

$G(1, \tau)$ is true immediately, since $M_1 = 1$. The ultimate goal is to prove $G(r+1, \tau)$ by induction.

Lemma 6. *For $i = 1, 2, \ldots, r$ and $\tau \geq 2$ is an integer, $[G(i, \tau) \cap P(i, \tau)] \rightarrow G(i+1, \tau)$*

Proof. Given the assumption, we have both $G(i, \tau)$ and $P(i, \tau)$ to be true. So for all $A \in Q_i$, $f(A) \geq \frac{1}{(\frac{2\tau-1}{\tau-1})^{i-1} M_i} \mathrm{OPT}_{M_i} \geq \frac{1}{(\frac{2\tau-1}{\tau-1})^{i-1} M_i} \mathrm{OPT}_{M_{i+1}}$ according to Lemma 5. For the next step, we need to prove $f(A_j) \geq \frac{\tau-1}{(2\tau-1)m_i} f(A)$ for all $A_j \in child(A) = \{A_1, A_2, \ldots, A_{m_i}\}$.

W.l.o.g, we assume $m_i \in \mathrm{argmin}_{j=1,2,\ldots,m_i} f(A_j)$ and $|A_{m_i}| = |\bar{A}_{m_i}| = k_{i+1}$. For $j = 1, 2, \ldots, m_i - 1$, we first look at $\bar{A}_j$. Since $P(i, \tau)$ is true, we know that $|\bar{A}_j| \geq \tau \geq 2$. Let a_j be the last added element in $\bar{A}_j$. We claim $f(\bar{A}_j \setminus a_j) \leq f(A_{m_i})$ given the greedy process and note that block A_{m_i} is not full (size smaller than k_{i+1}) at the time of adding a_j. Also, all elements in $\bar{A}_j$ are added by greedy order, so $f(\bar{A}_j) \leq \frac{|\bar{A}_j|}{|A_j|-1} f(\bar{A}_j \setminus a_j) \leq \frac{\tau}{\tau-1} f(A_{m_i})$ for $j = 1, 2, \ldots, m_i - 1$.

Next, there are still some elements in $A_j \setminus \bar{A}_j$ we have not analyzed. We notice $|\cup_{j=1,2,\ldots,m_i-1} [A_j \setminus \bar{A}_j]| \leq |A| - (m_i - 1)\tau - k_i$, and the elements in $\cup_{j=1,2,\ldots,m_i-1} [A_j \setminus \bar{A}_j]$ are not added to m_i block because they have less or equal marginal gains than any element in A_{m_i}. Therefore $f(A_{m_i} \cup \left[\cup_{j=1,2,\ldots,m_i-1} (A_j \setminus \bar{A}_j)\right]) \leq \frac{|A|-(m_i-1)\tau}{k_i} f(A_{m_i})$.

$$
\begin{aligned}
f(A) &\leq f(A_{m_i} \cup \left[\cup_{j=1,2,\ldots,m_i-1} (A_j \setminus \bar{A}_j)\right]) \\
&\quad + f(\cup_{j=1,2,\ldots,m_i-1} f(\bar{A}_j)) && (3.46) \\
&\leq \left[\frac{|A| - (m_i - 1)\tau}{k_i} + \frac{\tau(m_i - 1)}{\tau - 1}\right] f(A_{m_i}) && (3.47) \\
&= \left[\frac{m_i k_i - (m_i - 1)\tau}{k_i} + \frac{\tau(m_i - 1)}{\tau - 1}\right] f(A_{m_i}) && (3.48)
\end{aligned}
$$

$$\leq \left[1 + \frac{\tau}{\tau - 1}\right] m_i f(A_{m_i}) \tag{3.49}$$

$$\tag{3.50}$$

Therefore, we have $f(A_{m_i}) \geq \frac{1}{\frac{2\tau-1}{\tau-1}m_i} f(A)$. And since, $m_i \in \text{argmin}_{j=1,2,\ldots,m_i} f(A_j)$, we have $\min_{B \in child(A)} f(B) \geq \frac{1}{\frac{2\tau-1}{\tau-1}m_i} f(A)$ for all $A \in Q_i$. Recall that, we have, for all $A \in Q_i$, $f(A) \geq \frac{1}{(\frac{2\tau-1}{\tau-1})^{i-1}M_i}\text{OPT}_{M_{i+1}}$.

Combining them together gives $G(i+1, \tau)$. □

Finally, we finish the proof of Theorem 10 by induction, $[\cap_{i=1,2,\ldots,r} P(i, \tau)] \rightarrow G(r + 1, \tau)$. □

3.3 Round-Robin Greedy Based Algorithms

Barman and Krishna Murthy [9] proposes a round-robin style algorithm for the unconstrained robust submodular partition problem (Eq. (3.1)) and gives a constant bound of $\frac{1-e^{-1}}{3}$ in weakly polynomial running time. Compared to Min-Block Greedy, Round-Robin Greedy requires guessing the optimal values by an exponentially decreasing sequence, and for each guessed value, it runs one instance of round-robin subroutine. Specifically, suppose $\mu = \min_{j \in [m]} f(O_j)$, i.e., μ is the optimal solution value for the unconstrained case, then for a parameter $\delta > 0$, Round-Robin Greedy runs the round-robin subroutine with the guessed optimal values from a sequence $(f(V), \frac{f(V)}{1+\delta}, \frac{f(V)}{(1+\delta)^2}, \ldots)$ and ends when the guessed value is no larger than μ. The running time of each round-robin subroutine is $\mathcal{O}(n^2)$, as it greedily finds the element with the largest gain by iterating over all the remaining elements. There are $\log_{1+\delta} \frac{f(V)}{\mu}$ guessed values in the exponentially decreasing sequence, so the overall running time is $\mathcal{O}(n^2 \log_{1+\delta} \frac{f(V)}{\mu}) = \mathcal{O}(n^2 \frac{\log \frac{f(V)}{\delta\mu}}{\log 1+\delta}) = \mathcal{O}(n^2 \frac{1}{\delta} \log \frac{f(V)}{\mu})$. Note that since we use a $(1+\delta)$ factored exponentially decreasing sequence, we thus lose a $(1+\delta)$ factor in the approximation bound, which can be improved arbitrarily by using a smaller δ value at the cost of running more instances of the round-robin subroutine.

The major idea behind Round-Robin Greedy comes from the solution of Continuous

Greedy for the submodular welfare problem (homogeneous case), which is an uniform fractional vector $x = (\frac{1}{m}, \frac{1}{m}, \ldots, \frac{1}{m})$ with the length of x equal to the size of the expanded ground set $|\bar{V}|$. Let F be the multilinear extension [16] of f, Continuous Greedy gives a bound that $F(x) = \mathbb{E}_{R\sim x} f(R) \geq (1 - e^{-1}) \max_{\pi\in\Pi(V,m)} \sum_{A\in\pi} f(A)$, where $\mathbb{E}_{R\sim x} f(R)$ takes the expectation of $f(R)$ on a random set R with each element sampled independently according to the probability in the fractional vector x. Note that the hardness for submodular optimization under a matroid constraint is $1 - e^{-1}$, which means that the random assignment strategy achieves the best possible theoretical bound on the submodular welfare problem.

Round-Robin Greedy can be thought as a rounding mechanism for the fractional solution x. Intuitively, the round-robin style iteration is similar to the uniform random assignment in a deterministic manner, and by greedily finding the element, the value of every block can be bounded against that for the random assignment. In fact, Round-Robin Greedy bounds every block A_j to be $f(A_j) \geq \frac{1}{3}\frac{F(x)}{m}$, and since the welfare solution bounds the robust solution in terms of the sum: $\max_{\pi\in\Pi(V,m)} \sum_{A\in\pi} f(A) \geq \sum_{j\in[m]} O_j \geq m\mu$, we get the desired bound for the robust partition problem.

We extend Round-Robin Greedy to the constrained case (Eq. (3.10)) firstly with $\mathcal{C}$ as a cardinality constraint k. This is a relatively simple case due to the nature of Round-Robin Greedy that every block gets assigned with the same number of elements at the end of every round-robin iteration. We present the modified algorithm in Alg. 11, which also helps to explain the essential ideas of the original Round-Robin Greedy as we describe below.

Lemma 7 (**Cardinality Constrained Round-Robin**). *For the problem in Eq.* (3.10)*, with* $\mathcal{C}$ *as a cardinality constraint* k*, Alg. 11 gives a solution* $\min_{j\in[m]} f(A_j) \geq \frac{(1-e^{-1})^2}{3} \min_{j\in[m]} f(O_j^k)$.

Before we get into the proofs for the algorithm bounds, we will state the following lemma, which is a general property about robust submodular partitioning.

Lemma 8 (**Removal of one element and one block**). *For any* $v \in V$*, we have:*

$$\max_{\pi\in\Pi(V\setminus v,m-1,\mathcal{M})} \min_{A\in\pi} f(A) \geq \max_{\pi\in\Pi(V,m,\mathcal{M})} \min_{A\in\pi} f(A). \tag{3.51}$$

I.e., if we remove one element and one block from the problem, the optimal solution gets no worse.

Proof. Denote the optimal solution on V and m by $O_1, O_2, \ldots, O_m$ with $f(O_1) \leq f(O_2) \leq \ldots \leq f(O_m)$.

Suppose $v \in O_j$ for some j, then the blocks other than O_j forms a solution for problem defined on $V \setminus v$ and $m-1$, and we can add elements in $O_j \setminus v$ to other blocks (if the constraints permit). In the worst case, even if we cannot add any elements of $O_j \setminus v$ to other blocks, we still have $\max_{\pi \in \Pi(V,m,\mathcal{M})} \min_{A \in \pi} f(A) \geq \min_{j' \in [m], j' \neq j} f(O_{j'}) \geq f(O_1)$.

Suppose $\forall j \in [m],\ v \notin O_j$, then we only remove one block, and we can add the elements in that block to any other block so the solution value gets improved.

□

For Round-Robin Greedy based algorithms, we first guess the optimal solution value and then assign singletons to blocks which satisfies the bound based on the guessed optimal value. After that, we run the algorithm on the restricted problem with those blocks and elements removed. By applying the previous lemma (recursively if multiple elements and blocks removed), we know that the optimal solution on the restricted instance is no worse than the optimal solution on the original problem. Therefore, it suffices to analyze the solution on the restricted instance. Next, we show the proof for Lemma 7;

Proof. By solving $\max_{\pi \in \Pi(V',m',k)} \sum_{S \in \pi} f(S)$ in Line 12 of Alg. 11 (Theorem III.3 in [20]), we know that

$$\sum_{j \in [m]'} f(A'_j) \geq (1 - e^{-1}) \max_{\pi \in \Pi(V',m',k)} \sum_{S \in \pi} f(S). \tag{3.52}$$

Recall that we denote the optimal solution value in the cardinality constraint case by $OPT^{\mathcal{M}^u_k}$, where k is the cardinality. We assumes we know the optimal solution value $OPT^{\mathcal{M}^u_k}$ for this proof. For the algorithm, the $OPT^{\mathcal{M}^u_k}$ value is guessed within a factor of $\frac{1}{1+\delta}$. Therefore, to be more precise, we have an additional factor of $\frac{1}{1+\delta}$ in the bound, which can be made arbitrarily small by setting δ small.

For the limited ground set V'', running unconstrained round-robin ensures every block to have at most k elements and therefore the cardinality constraint is satisfied. Suppose we run the continuous greedy algorithm on the limited ground set V'' with the submodular welfare objective ($\max_{\pi \in \Pi(V'',m')} \sum_{S\in\pi} f(S)$), we get a fractional solution $x_1 = x_2 = \ldots = x_{m'} = (\frac{1}{m'}, \frac{1}{m'}, \ldots, \frac{1}{m'})$ (we do not really need to run the algorithm, but we will compare our solution to the fractional solution). Denote the multilinear extension of f by F and $F(x) = \mathbb{E}_{R\sim x} f(R)$ (we can think it as the expected value of f where every element is sampled independently based on probabilities defined in vector x). Consider any block A_j in the solution of Alg 11 for $j \in [m']$ (for $j \notin [m']$, those blocks are the singleton assignment blocks and they satisfy the bound by construction), we have:

$$\frac{(1-e^{-1})^2}{3} OPT^{\mathcal{M}_k^u} + 2f(A_j) \geq F(x_j) \tag{3.53}$$

$$\geq \frac{1-e^{-1}}{m'} \max_{\pi\in\Pi(V'',m')} \sum_{S\in\pi} f(S) \tag{3.54}$$

$$\geq \frac{1-e^{-1}}{m'} \max_{\pi\in\Pi(V'',m',k)} \sum_{S\in\pi} f(S) \tag{3.55}$$

$$\geq \frac{(1-e^{-1})^2}{m'} \max_{\pi\in\Pi(V',m',k)} \sum_{S\in\pi} f(S) \tag{3.56}$$

$$\geq (1-e^{-1})^2 \max_{\pi\in\Pi(V',m',k)} \min_{S\in\pi} f(S). \tag{3.57}$$

$$\geq (1-e^{-1})^2 OPT^{\mathcal{M}_k^u}. \tag{3.58}$$

Rearrange and we get:

$$f(A_j) \geq \frac{(1-e^{-1})^2}{3} OPT^{\mathcal{M}_k^u}. \tag{3.59}$$

Eq. (3.53) comes from the Lemma.3 of [9], in which case we can bound every block in the round-robin solution to the fractional solution of the continuous greedy algorithm on the multilinear extension of f. Note for Lemma.3 of [9], they study the unconstrained case and show that $\frac{\gamma}{3} OPT^{\mathcal{M}_k^u} + 2f(A_j) \geq F(x_j)$ with $\gamma = 1 - e^{-1}$. γ comes from the singleton assignment step, which assigns blocks with singletons whose values are larger than $\frac{\gamma}{3} OPT^{\mathcal{M}_k^u}$.

A slightly more general statement can be made for any $0 \leq \gamma \leq 1$ with the same proof as Lemma.3 of [9]. In our case, we pick $\gamma = (1 - e^{-1})^2$. Eq. (3.54) follows from the property of the continuous greedy solution. The continuous greedy gives a $(1 - e^{-1})$ approximation to the submodular welfare problem, and the fractional solution x_j for each block is the same. Therefore, every block's evaluation in expectation is at least $\frac{(1-e^{-1})}{m'}$ of the submodular welfare optimal solution. Eq. (3.55) follows that the unconstrained solution is no worse than the constrained solution. Eq. (3.56) uses Eq. (3.52): $A'_1, \ldots, A'_m$ is one possible solution to $\max_{\pi \in \Pi(V'', m', \mathcal{M}_k^u)} \sum_{S \in \pi} f(S)$, and we know $\sum_j f(A'_j) \leq \max_{\pi \in \Pi(V'', m', k)} \sum_{S \in \pi} f(S)$ because of the max operator. Therefore, we have $\max_{\pi \in \Pi(V'', m', k)} \sum_{S \in \pi} f(S) \geq \sum_j f(A'_j) \geq (1 - e^{-1}) \max_{\pi \in \Pi(V', m', k)} \sum_{S \in \pi} f(S)$. Eq. (3.57) follows that the sum over blocks of the max-min solution is no larger than the optimal welfare solution. Eq. (3.58) uses Lemma. 8: the optimal max-min solution on V' and m' is no worse than the optimal max-min solution on V and m.

As stated above, $\frac{\gamma}{3} OPT + 2f(A_j) \geq F(x_j)$ is true for any $0 \leq \gamma \leq 1$. It may seem that making γ smaller can improve the bound. However, we only bound the m' blocks but not the singleton blocks. Because of the singleton assignment step, every singleton has a value larger than $\frac{\gamma}{3}$, and the final bound over all m blocks will be the minimal of the bound on the m' blocks and the singleton blocks. Setting γ small worsens the bound on the singleton blocks. To balance the two bounds, $\gamma = (1 - e^{-1})^2$ is picked so that the bounds on the m' blocks and the singleton blocks meet.

□

Different from the original Round-Robin Greedy, which performs a grid search of the guessed optimal values, we perform a binary search over the sequence of values and therefore the number of outer iterations is reduced. Moreover, we use the Min-Block Greedy solution's value as the minimum guessed value τ. Because of the $\frac{1}{m + \frac{1}{1-e^{-1}}}$ bound of the Min-Block Greedy solution, the maximum guessed value is thus bounded by $(m + 2)\tau$. We then create a $1 + \delta$ factored exponential decreasing sequence between τ and $(m + 2)\tau$ to binary search the optimal solution value. This improves the number of outer iterations of the algorithm

to $\mathcal{O}(\log\log_{1+\delta} m) = \mathcal{O}(\log\log m + \log\frac{1}{\delta})$, which is strongly-polynomial while the number of outer iterations $\mathcal{O}(\log_{1+\delta}\frac{f(V)}{\mu})$ for the original unconstrained case is weakly-polynomial as it has a log dependence on the function value.

In every outer iteration (Line 22-25), Alg 11 checks whether the round-robin solution based on the guessed optimal value $(1+\delta)^{idx}\tau$ satisfy the approximation bound, i.e., $f(A_j) \geq \frac{(1-e^{-1})^2}{3}(1+\delta)^{idx}\tau \ \forall j \in [m]'$. If the bound is (not) satisfied, the guessed value is large (small) and we move to an increased (decreased) binary search value. Within every outer iteration, we perform a round-robin style greedy, where we iterate over every block in some fixed order and greedily add to the block an element contributing the largest gain. Line 8 of Alg. 11 is the major change to Round-Robin Greedy specifically for the cardinality constraint case, where we first find the solution to the cardinality constrained submodular welfare problem $\max_{\pi\in\Pi(V',m',k)}\sum_{S\in\pi} f(S)$, and then only apply Round-Robin Greedy to the union V'' of the solution $A'_1, A'_2, \ldots, A'_{m'}$.

The running time of Alg. 11 is similar to Round-Robin Greedy, with additional costs induced by Line 12, which solves a cardinality constrained submodular welfare problem. Using Continuous Greedy and swap rounding [20] for Line 12 can be quite costly ($\mathcal{O}(n^5)$ for the inner loop), which may be improved in future research by a more efficient algorithm. In Alg. 12, we propose another algorithm, which addresses the constrained robust submodular problem with $\mathcal{C}$ as any matroid constraint $\mathcal{M}$ and incurs no additional computation costs compared to Round-Robin Greedy.

Theorem 11 (**Matroid Constrained Round-Robin**). *For the problem in Eq.* (3.10)*, with* $\mathcal{C}$ *as any matroid constraint* $\mathcal{M}$*, Alg 12 gives a solution* $\min_{j\in[m]} f(A_j) \geq \frac{(1-e^{-1})}{5}\min_{j\in[m]} f(O_j^{\mathcal{M}})$.

Before we get in the proof of Theorem 11, we first need to prove the following lemma about the property of the continuous greedy solution.

Lemma 9 (**Continuous Greedy Solution**). *For the constrained welfare problem*

$$\max_{\pi \in \Pi(V,m,\mathcal{M})} \sum_{A \in \pi} f(A),$$

the continuous greedy algorithm outputs a fractional solution $x_1 = x_2 = \ldots = x_m$ *(*$x_j \in [0,1]^n$*), which is the same for every block in the partition and*

$$\sum_{j \in [m]} F(x_j) \geq (1 - e^{-1}) \max_{\pi \in \Pi(V,m,\mathcal{M})} \sum_{A \in \pi} f(A).$$

F *is the multilinear extension of* f*, i.e.,* $F(x) = \mathbb{E}_{R \sim x} f(R)$ *(we can think it as the expected value of* f *where every element is sampled independently based on probabilities defined in vector* x*). Moreover,* $\forall i \in [n], j \in \{1, \ldots, m\}, x_j[i] \leq \frac{1}{m}$ *and* $\sum_{i \in V} x_j[i] \leq r_{\mathcal{M}}(V)$.

Proof. Note that the continuous greedy can give a fractional solution with $1 - e^{-1}$ bound under any solvable polytope constraint. It's the rounding procedure that limits the constraint we can use to get a set solution, e.g., with pipage rounding, we can use any matroid constraint.

In fact, we do not need to run the continuous greedy algorithm, and we only need to show the existence of a solution. Suppose the solution to the $max_y\{w \cdot y, y \in \mathcal{P}\}$ step of the continuous greedy algorithm is given by some oracle. Given the direction y, we just evenly split the resulting vector y among the m blocks, as we cannot distinguish between blocks. At the end of the algorithm, we will have the fractional solution $x_1 = x_2 = \ldots = x_m$ and $\sum_{j \in [m]} F(x_j) \geq (1 - e^{-1}) \max_{\pi \in \Pi(V,m,\mathcal{M})} \sum_{A \in \pi} f(A)$.

Since the fractional solution are guaranteed to be in the matroid polytope of $\mathcal{M}$ and $\mathcal{M}_m^p$, we have $\forall i \in [n], j \in \{1, \ldots, m\}, x_j[i] \leq \frac{1}{m}$ and $\sum_{i \in V} x_j[i] \leq r_{\mathcal{M}}(V)$.

□

The general idea of proving Theorem 11 is to bound the solution to the fractional solution of the continuous relaxation. For every block in the solution, we inspect the elements that have been evaluated during the greedy step. For those elements with large gains when being evaluated but not added due to the violation of the matroid constraint, we bound their gains as submodular maximization with a matroid constraint on a reduced ground set. For the

remaining elements, we bound their gains by the greedy step and together we get the desired bound. Next, we show the proof of the bound in a restricted setting where no large singleton elements are present in the problem instance.

Lemma 10 (**Matroid Constraint Round-robin with No Large Singletons).** *Suppose for all $v \in V$, we have $f(v) \leq \frac{1-e^{-1}}{5} OPT^{\mathcal{M}}$, where $OPT^{\mathcal{M}}$ is the optimal solution value of the robust submodular partition problem constrained by matroid $\mathcal{M}$ (in other words, all the singletons have relatively small values for the given problem instance). Then, the round-robin iterations of Alg. 12 (line 14-28) gives a solution* $\min_{j\in[m]} f(A_j) \geq \frac{1-e^{-1}}{5} OPT^{\mathcal{M}}$.

Proof. Let's focus on one block (any one in $A_1, \ldots, A_{m'}$) and for simplicity, we will omit the block index j for this proof if not further noticed. Denote $OPT = \min_{j\in[m]} f(O_j^{\mathcal{M}})$ for this proof. Also, we assumes we know the optimal solution value OPT for this proof. Note that in the complete version of Alg. 12, we need to remove large singleton values based on the guessed optimal value, but for this lemma we make the assumption that in the given problem instance, there are no large singletons present.

For the current block, we denote the final resulting set from Alg. 12 as A. For one round-robin iteration, we go over all the feasible blocks sequentially, and to get A we need to run $|A| = r$ round-robin iterations. Note that for different blocks, the number of round-robin iterations might be different.

We then divide the restricted ground set V' by the round-robin iterations with respect to the current block A. Before we add the first element to A, denote all the allocated elements by V^0. Then we can think that for every round-robin iteration, we always start from the current block A. Let $V' = V^0 \cup V^1 \cup \ldots \cup V^r$ be a partition of V' and V^t contains all the elements allocated during the t's round-robin iteration. Note V^r contains all the unallocated elements in the ground set after we add the last element to A. Let $V^{t_1:t_2} = \cup_{t\in\{t_1,t_1+1,\ldots,t_2\}} V^t$. Accordingly, we partition the result A by $A^t = A \cap V^t$.

For the set $V' \setminus A$, we separate it into two parts Q_1' and Q_2', where Q_1' contain all the elements checked in Alg. 12 that cannot be added to the current block due to the matroid

constraint, and let $Q_2' = V' \setminus A \setminus Q_1'$, i.e., Q_2' contain all the elements that can be added the current block. To be more precise:

$$Q_1' = \cup_{t\in\{0,1,\ldots,r\}} \cup_{v\in V^t\setminus A^t,(A^{1:t}\cup v)\notin\mathcal{M}} v \tag{3.60}$$

Let $Q_1 = Q_1' \cup A$ and $Q_2 = Q_2' \cup A$.

Let F denote the multilinear extension of f, i.e., $F(x) = \mathbb{E}_{R\sim x} f(R)$. By Lemma 8 and Lemma 9, we know that

$$(1-e^{-1})OPT \leq \max_{\pi\in\Pi(V',m',\mathcal{M})} \min_{S\in\pi} f(S) \tag{3.61}$$

$$\leq \frac{1}{m'} \max_{\pi\in\Pi(V',m',\mathcal{M})} \sum_{S\in\pi} f(S) \tag{3.62}$$

$$\leq F(x) \tag{3.63}$$

$$\leq (F(x\cap Q_1) + F(x\cap Q_2)). \tag{3.64}$$

Note that x is the fractional solution to the continuous greedy algorithm on the welfare objective: $\max_{\pi\in\Pi(V',m',\mathcal{M})} \sum_{S\in\pi} f(S)$ (similar to one of the x_j's in Lemma 9 and we omit the block index for this proof). Here we use $x \cap Q$ to represent setting all elements not in Q to be zero in the x fractional solution. The first inequality follows from Lemma. 8. The second inequality follows that the sum over blocks of the max-min solution is no better than the optimal solution of the welfare problem. Since every element is sampled independently according to its probability in the fractional solution x, together with submodularity ($Q_1 \cup Q_2 = V'$) we get the last inequality above. Next, we will bound the two terms $F(x \cap Q_1)$ and $F(x \cap Q_2)$ separately.

For the first term $F(x \cap Q_1)$, we know that $r = r_{\mathcal{M}}(Q_1)$, and Alg. 12 generates A in the same manor as running greedy max on Q_1 with matroid constraint $\mathcal{M}$. To be more precise, suppose $\mathcal{M} = (V, \mathcal{I})$ and we remove all the elements that are not in Q_1 and get $\mathcal{M}' = (V' \cap Q_1, \{I \cap Q_1 \forall I \in \mathcal{I}\})$. Note that $\mathcal{M}'$ is also a matroid, and all sets that satisfy $\mathcal{M}'$ also satisfy $\mathcal{M}$ due to the down-monotone property of matroids. Therefore, we have:

$$f(A) \geq \frac{1}{2} \max_{S\in\mathcal{M}'} f(S) \tag{3.65}$$

$$\geq \frac{1}{2}F(x \cap Q_1). \tag{3.66}$$

Note that x is in the matroid polytope of $\mathcal{M}$, and $x \cap Q_1$ is in the matroid polytope of $\mathcal{M}'$. By pipage rounding, we know that we can get an integral solution X' from $F(x \cap Q_1)$ so that the integral solution still satisfies $X' \in \mathcal{M}'$ and $f(X') \geq F(x \cap Q_1)$. Since $\max_{S \in \mathcal{M}'} f(S) \geq f(X')$, we get the last inequality above.

For the second term $F(x \cap Q_2)$, we will bound it using the greedy step. Denote $y = x \cap Q_2$, $y^t = y \cap V^t$, and $\mathbb{E}_S(y) = \mathbb{E}_{R \sim y} f(R|S)$, we have:

$$\begin{aligned}
F(y) &= \mathbb{E}_{R \sim y} f(R) && (3.67)\\
&= \mathbb{E}_{R \sim y^0} f(R) + \mathbb{E}_{R^1 \sim y^0, R^2 \sim y^{1:r}} f(R^2|R^1) && (3.68)\\
&\leq F(y^0) + F(y^{1:r}) && (3.69)\\
&\leq F(y^0) + f(A^1) + \mathbb{E}_{A^1}(y^{1:r}) && (3.70)\\
&= F(y^0) + f(A^1) + \mathbb{E}_{A^1}(y^1) + \mathbb{E}_{R^1 \sim y^1, R^2 \sim y^{2:r}} f(R^2|R^1 \cup A^1) && (3.71)\\
&\leq F(y^0) + f(A^1) + \mathbb{E}_{A^1}(y^1) + \mathbb{E}_{A^1}(y^{2:r}) && (3.72)\\
&\leq F(y^0) + f(A^1) + \mathbb{E}_{A^1}(y^1) + f(A_2|A_1) + \mathbb{E}_{A^2}(y^{2:r}) && (3.73)
\end{aligned}$$

Continue to unwrap $\mathbb{E}_{A^2}(y^{2:r})$ in the same way, finally we get:

$$\begin{aligned}
F(y) \leq\ & F(y^0) + \left[f(A^1) + f(A^2|A^1) + f(A^3|A^2) + \ldots + f(A^r|A^{r-1})\right] \\
& + \left[\mathbb{E}_{A^1}(y^1) + \mathbb{E}_{A^2}(y^2) + \ldots + \mathbb{E}_{A^r}(y^r)\right] && (3.74)\\
=\ & F(y^0) + f(A) + \left[\mathbb{E}_{A^1}(y^1) + \mathbb{E}_{A^2}(y^2) + \ldots + \mathbb{E}_{A^r}(y^r)\right] && (3.75)
\end{aligned}$$

We then need to bound $F(y^0)$ and $[\mathbb{E}_{A^1}(y^1) + \mathbb{E}_{A^2}(y^2) + \ldots + \mathbb{E}_{A^r}(y^r)]$. Note that because of the OPT guessing process, we remove all the singleton gains larger than $\frac{1-e^{-1}}{5}$, and we have:

$$F(y^0) \leq \frac{1 - e^{-1}}{5} OPT \tag{3.76}$$

Since we select items greedily at every round-robin step, and y only has non-zero values for elements that are in Q_2, we have:

$$\mathbb{E}_{A^t}(y^{t+1}) = \mathbb{E}_{R \sim y^{t+1}} f(R|A^t) \tag{3.77}$$

$$\leq \sum_{v \in y^{t+1}} y^{t+1}(v) f(v|A^t) \tag{3.78}$$

$$\leq \sum_{v \in y^{t+1}} \frac{1}{m'} f(v|A^t) \tag{3.79}$$

$$\leq f(A^t|A^{t-1}) \tag{3.80}$$

Note that for the last round-robin iteration V^r, it may seem that there can be more than m' elements, but it's not possible: since there are no new elements added to the current blocks, $V^r \cap Q_2$ contains at most m' elements as otherwise we will find new feasible elements and add to block A.

Then we sum over all t and get:

$$\left[\mathbb{E}_{A^1}(y^1) + \mathbb{E}_{A^2}(y^2) + \ldots + \mathbb{E}_{A^r}(y^r)\right] \leq f(A). \tag{3.81}$$

Therefore, we have:

$$(1 - e^{-1})OPT \leq (F(x \cap Q_1) + F(x \cap Q_2)) \tag{3.82}$$

$$\leq 2f(A) + F(y) \tag{3.83}$$

$$\leq 2f(A) + 2f(A) + \frac{1 - e^{-1}}{5} OPT \tag{3.84}$$

$$f(A) \geq \frac{1 - e^{-1}}{5} OPT. \tag{3.85}$$

□

Next, we will discuss why binary search can be used in the guessing of the optimal value (as opposed to the case of linear search where we try all possible guessed optimal values). We will take Alg. 12 as an example, and the same argument follows for Alg. 11.

Lemma 11 (**Binary Search**). *For Alg. 12, let the potential guessed optimal values form a sequence* $(\tau_1, \tau_2, \ldots, \tau_l)$ *where* $\tau_{i+1} = (1 + \delta)\tau_i$. *Then for any* $\tau_i \leq OPT^{\mathcal{M}}$ *as the guessed optimal value that we plug into the round-robin iterations (Alg. 12 line 7-28), line 24 of Alg. 12 is always true.*

Proof. For simplicity, denote the optimal solution of the matroid constrained robust partitioning problem as OPT for this proof. Let's denote the found large singleton values and the remaining sets (line 8 and 9 of Alg. 12) respectively by G_{OPT} and V'_{OPT} for OPT, and G_i and V'_i for τ_i. Since $\tau_i \leq OPT$, $G_{OPT} \subseteq G_i$ as the threshold is smaller for τ_i. Then by Lemma. 8, we know that the optimal solution OPT_i on V'_i (partitioned to $m - |G_i|$ blocks) is no less than the optimal solution OPT' on V'_{OPT} (partitioned to $m - |G_{OPT}|$ blocks). Also note that since we remove all singleton values to form V'_i based on the threshold $\frac{1-e^{-1}}{5}\tau_i$, and $\tau_i \leq OPT \leq OPT' \leq OPT_i$ ($OPT \leq OPT'$ is also from Lemma. 8), we are guaranteed that there are no singleton elements with $f(v) \geq \frac{1-e^{-1}}{5}OPT_i$ in V'_i. Therefore, based on Lemma. 10, our round robin iterations on V'_i give a solution whose every block has a value of at least $\frac{1-e^{-1}}{5}OPT_i \geq \frac{1-e^{-1}}{5}\tau_i$, and thus line 24 is guaranteed to be true for τ_i. □

Based on Lemma. 11, either of the following must hold: 1) for all $\tau_i > OPT^{\mathcal{M}}$ line 24 is false, and in such a case, we can use binary search to find the largest τ_i with line 24 true, and let's call it τ_{i^*}; 2) there exists some $\tau_i > OPT^{\mathcal{M}}$ such that line 24 is true. If we find such τ_i, we find a solution with $f(A_j) \geq \frac{1-e^{-1}}{5}\tau_i \geq \frac{1-e^{-1}}{5}OPT^{\mathcal{M}}$. Otherwise if we don't find it, we go to the first case and will still find τ_{i^*}.

Finally, the approximation guarantee of Alg.12 follows by combining the previous lemmas.

Theorem 11 (**Matroid Constrained Round-Robin**). *For the problem in Eq.* (3.10)*, with* $\mathcal{C}$ *as any matroid constraint* $\mathcal{M}$*, Alg 12 gives a solution* $\min_{j\in[m]} f(A_j) \geq \frac{(1-e^{-1})}{5} \min_{j\in[m]} f(O_j^{\mathcal{M}})$.

Proof. Firstly, based on the previous arguments about the binary search, we can find a τ_i with $\tau_i \geq \tau_{i^*}$, where τ_{i^*} is the largest τ_j with $\tau_j \leq OPT^{\mathcal{M}}$ and line 24 of Alg. 12 true. By setting δ small, τ_{i^*} can be arbitrarily close to $OPT^{\mathcal{M}}$. Next, based on Lemma. 10, after removing the large singleton values, the solution on the remaining elements have the min block value at least $\frac{1-e^{-1}}{5}\tau_i$, and the removed large singletons are all larger than $\frac{1-e^{-1}}{5}\tau_i$ by construction. We therefore get the approximation ratio. □

Comparing to Alg. 11, the major change in Alg. 12 is (1) we do not need to run the costly Continuous Greedy and swap rounding to get a solution to the constrained welfare

problem, which makes the algorithm applicable in practice; (2) for Line 15-23, we find a feasible element and add it to the current block, and we remove a block from the candidate set J if there are no element in the remaining set that can be added to the block without violating the matroid constraint. The overall running time is $\mathcal{O}(n^2(\log\log m + \log\frac{1}{\delta}))$.

Algorithm 11: Cardinality Round-Robin Greedy

input : f, V, m, k, δ

Let τ be the solution value of Alg. 8;

Let $high = \lceil \log_{1+\delta}(m+2) \rceil$, $low = 0$;

Create a sequence of guessed values: $(\tau, (1+\delta)\tau, (1+\delta)^2\tau, \ldots, (1+\delta)^{high}\tau)$;

Create an empty solution ($\emptyset$ for each block in the partition) for each guessed value $\pi_0, \pi_1, \ldots, \pi_{high}$;

while $high \geq low$ **do**

  Let $idx = \lfloor (high + low)/2 \rfloor$;

  Let $A_1 = A_2 = \ldots = A_m = \emptyset$;

  Let $V' = \{v | v \in V, f(v) \leq \frac{(1-e^{-1})^2}{3}(1+\delta)^{idx}\tau\}$;

  Let $G = V \setminus V'$;

  Assign G to $A_{m-|G|+1}, A_{m-|G|+2}, \ldots, A_m$ with one element per block;

  Let $m' = m - |G|$;

  Let $A'_1, A'_2, \ldots, A'_{m'}$ be the solution to $\max_{\pi \in \Pi(V', m', k)} \sum_{S \in \pi} f(S)$ using continuous greedy and swap rounding;

  Let $V'' = \cup_{j \in [m]'} A'_j$;

  Let $R = V''$;

  for $i = 1 : k$ **do**

    for $j \in [m]'$ **do**

      Let $v^* \in \text{argmax}_{v \in R} f(v | A_j)$;

      $A_j := A_j \cup \{v^*\}$;

      $R := R \setminus \{v^*\}$;

  if $f(A_j) \geq \frac{(1-e^{-1})^2}{3}(1+\delta)^{idx}\tau \ \forall j \in [m']$ **then**

    Let $\pi_{idx} = \{A_1, A_2, \ldots, A_m\}$;

    Let $low = idx + 1$;

  else

    Let $high = idx - 1$;

return best of $\pi_0, \pi_1, \ldots, \pi_{high}$;

Algorithm 12: Matroid Round-Robin Greedy

input : f, V, m, $\mathcal{M}$, δ

Let τ be the solution value of Alg. 8;
Let $high = \lceil \log_{1+\delta}(m+2) \rceil$, $low = 0$;
Create a sequence of guessed values: $(\tau, (1+\delta)\tau, (1+\delta)^2\tau, \ldots, (1+\delta)^{high}\tau)$;
Create an empty solution ($\emptyset$ for each block in the partition) for each guessed value $\pi_0, \pi_1, \ldots, \pi_{high}$;
while $high \geq low$ **do**
    Let $idx = \lfloor (high + low)/2 \rfloor$;
    Let $A_1 = A_2 = \ldots = A_m = \emptyset$;
    Let $V' = \{v | v \in V, f(v) \leq \frac{1-e^{-1}}{5}(1+\delta)^{idx}\tau\}$;
    Let $G = V \setminus V'$;
    Assign G to $A_{m-|G|+1}, A_{m-|G|+2}, \ldots, A_m$ with one element per block;
    Let $m' = m - |G|$;
    Let $R = V''$;
    Let $J = [m']$;
    while $J \neq \emptyset$ *and* $R \neq \emptyset$ **do**
        for $j \in [m]'$ **do**
            if $j \in J$ **then**
                if $\exists v$ *s.t.* $A_j \cup \{v\} \in \mathcal{M}$ **then**
                    $v^* \in \underset{v \in R, A_j \cup v \in \mathcal{M}}{\operatorname{argmax}} f(v|A_j)$;
                else
                    Let $J = J \setminus j$;
                    continue;
                $A_j := A_j \cup \{v^*\}$;
                $R := R \setminus \{v^*\}$;
        if $f(A_j) \geq \frac{(1-e^{-1})}{5}(1+\delta)^{idx}\tau \ \forall j \in [m']$ **then**
            Let $\pi_{idx} = \{A_1, A_2, \ldots, A_m\}$;
            Let $low = idx + 1$;
        else
            Let $high = idx - 1$;
return best of $\pi_0, \pi_1, \ldots, \pi_{high}$;

Chapter 4

GENERATING FIXED MINI-BATCHES VIA SUBMODULAR ROBUST PARTITIONING

We further study the cardinality constrained robust partitioning problem and apply it to generate representative and fixed mini-batches for stochastic gradient methods. While sampling independent random mini-batches is essential from a theoretical perspective, it intrinsically conflicts with the efficient use of computational learning systems. Training sets are getting larger (thereby driving accuracy higher) and they typically do not fit in cache or memory. The only feasible approach is to repeatedly load data from main memory and/or disk to form mini-batches, but doing so from a convergence rate perspective (i.e., randomly with replacement) is costly because caches do not help when memory access patterns are random and hence unpredictable.

In general, to achieve high computational efficiency, time spent loading an independent minibatch should occur simultaneously with computation on a previous minibatch, as in a pipeline. Prepossessing techniques, such as data augmentation [25, 64, 128, 43], which have been developed to improve the accuracy, can mitigate demands on memory bandwidth since augmented data may be created via access only to local caches. While this reduces memory bandwidth requirements, at a cost of less independent minibatches, it is only a stopgap measure, as computational capability is improving faster than available memory bandwidth. For example, GPUs are more than five times faster than a few years ago, e.g., the Nvidia V100 (7.8 TFLOPs) vs. the K40 (1.4 TFLOPs), not to mention issues associated with having multiple hungry GPUs on the same machine running simultaneously. Hence, independent random sampling of mini-batches is becoming ever more impractical.

As sequential access is significantly faster than random access, the only practical strat-

egy is to iterate through a fixed sequence of data samples (written a priori to disk) rather than obtain an unbiased random mini-batch at every gradient update step. Although we could consider randomly re-shuffle the data points after every epoch, this can also be an overwhelmingly costly operation. Therefore, to achieve efficient training systems in practice, we often rely on one fixed sequence of data points to iterate through multiple times to train a model [145]. For example, when training a deep model on ImageNet [102], a commonly used approach is to generate a fixed randomly shuffled list of indices of the images, construct a database (usually optimized for sequential access) based on this list, and iterate through the order multiple times [57]. Even though applying stochastic gradient methods on a deterministic sequence of data can be theoretically suboptimal, the benefits include improved and predictable data access patterns and better reproducibility.

Ideally, a good deterministic sequence of mini-batches should have the following properties: (1) mini-batch representativeness, where every mini-batch is representative and non-redundant, and thus stochastic gradients calculated using the mini-batches are not too far from the true gradients; and (2) order consistency, where groups of mini-batches in the sequence should be more broadly representative, so the corresponding sequence of gradients does not drive a model in the wrong direction. As mentioned above, however, one widely used approach to generate the fixed sequence is random shuffling, which could result in a suboptimal sequence due to non-representativeness. This can impede the performance of the trained model.

We propose to use cardinality constrained submodular robust partitioning to generate a sequence of mini-batches given a set of data samples. Our method consists of hierarchical runs of max-min robust submodular partition of the dataset with various cardinality constraints, which generates a partial order over mini-batches of data points within a hierarchical structure, and where both mini-batches and groups of mini-batches in the hierarchy are encouraged to be representative of the entire dataset. On deep learning tasks, we show that our deterministic sequences of mini-batches significantly outperform the worst and mean of randomly generated sequences (both are likely to happen in practice), and for most of the

cases, beat the best of random sequences.

4.1 Related Work

For our application of using cardinality constrained robust submodular partitioning to generate mini-batches, we compare our approach with previous methods as follows. A mini-batch diversification method based on determinantal point processes (DPPs) is given in [150], which relies on similarity measurements between data points so that mini-batches with redundant data are given low-probability and mini-batches with more diverse data are given high-probability. Our objective is different from [150], as we aim to generate a fixed sequence of representative mini-batches for sequential disk access, while their method generates non-deterministic mini-batches. Moreover, their method does not enforce representativeness of groups of mini-batches, while our approach does. We also note that DPPs are related to submodular functions (a DPP is log-submodular), and our framework is very general as it can be used with any submodular function. Finally, DPP methods are computationally expensive, while our hierarchical partitioning method is efficient enough for very large datasets such as ImageNet. Salehi et al. [103] give a multi-arm bandit sampling approach to adaptively sample mini-batches for stochastic optimization with the aim to better control the variance of gradients of the mini-batches and to improve the convergence rate. This work focuses on generating random mini-batches with less variance than uniformly random during training (their method requires accessing gradient information), while we strive to find a fixed mini-batch sequence before training starts. Both [150] and [103] shares similar intuition with us that mini-batches with more representativeness/more diversity can contribute to better performance of the training system. Our approach, however, stays mindful of the computational requirements as well. [108] proposes a sampling strategy specifically for the sampling with-out replacement scenario, which exhibits convergence properties for convex problems but it requires a global (entire dataset) gradient calculation at each minibatch which can be very slow especially for DNNs. [155] modifies the stochastic algorithms to utilize the structural information obtained from raw clustering on the training dataset to get

improved running time. Again, it requires a global gradient calculation and is not designed for minibatch stochastic gradient settings. Curriculum learning [10] and self-paced learning with diversity [59] are also related as they focus on generating mini-batches better during the training process. However, they require random data access, which is often infeasible for very large data sets. Also, theses methods require to interact with the training model and produce minibatches adaptively on the fly. In contrast, our method produces a single reusable sequential high quality data access pattern, a key unique benefit of our method.

4.2 Cardinality Constraint Robust Partitioning for Generating Fixed Mini-batches

In this chapter, we study a special case of cardinality constrained robust submodular partitioning problem with the application of generating representative mini-batches for stochastic gradient methods. We will follow the same formulation and notation as defined in Chapter 3. Particularly, our objective is:

$$\max_{\pi \in \Pi(V,k)} \min_{i=1:m} f(\pi_i(V)). \tag{4.1}$$

We will focus on the Min-block greedy algorithm for this section due to the better running-time of the Min-block greedy algorithm.

For our mini-batch partitioning task, we focus on generating a sequence of blocks such that every block in the partition is a mini-batch to train on. Suppose every resulting mini-batch is representative due to the max-min robust objective, then the ordering of the mini-batches used to train becomes less crucial, and an arbitrary ordering of the mini-batches will suffice. However, for most problems, the mini-batches are too small to represent the ground set V, in which case the ordering of the mini-batches can be critical, as groups of mini-batches can be redundant even though every mini-batch is as representative and non-redundant as possible for its size. For now, however, we assume the simple case where the mini-batch size is large enough to represent the ground set.

The robust submodular partitioning objective naturally describes our requirements about a deterministic mini-batch sequence of training data. Firstly, the max-min objective ensures

that all mini-batches are as representative as possible of the entire training dataset. Secondly, the class of submodular functions we can use for the objective is sufficiently expressive that by solving such a problem, we have a general framework that could potentially work for various forms of data and different definitions of representativeness. We choose to use the Min-Block Greedy based algorithm (Alg. 8) to optimize our target. Recall we prove the following approximation guarantee for our objective:

Theorem 9 (**Min-Block Cardinality Constraint).** *For submodular function f on ground set V and block size (mini-batch size) constraint k, suppose $m = |V|/k$, Algorithm 8 gives an approximation ratio of $\frac{e-1}{(e-1)m+1}$.*

The worst case running time of Algorithm 8 is $\mathcal{O}(|V|^2)$ evaluations of f plus $\mathcal{O}(|V|^2)$ basic operations. We care mostly about the number of function evaluations as f can be significantly more expensive than the basic operations. Though not wholly intractable, $\mathcal{O}(|V|^2)$ evaluations can become overwhelming when the dataset size gets large. To accelerate the algorithm, we can apply the lazy evaluation trick [80] at line 5 of Algorithm 8. Essentially, we can create a priority queue for each of the m blocks, and initialize every priority queue with all the singleton values $f(v), v \in V$. When we pop the top of the priority queue, get element v and evaluate $f(v|A_{j^*})$, and if such value is larger than the next value in the priority queue, then by submodularity, v is guaranteed to be in $argmax_{v \in R} f(v|A_{j^*})$, or otherwise we push $f(v|A_{j^*})$ back to the priority and keep popping. In the worst case, the running time with the priority queue would still be $\mathcal{O}(|V|^2)$ evaluations of f, however, it is widely recognized that in practice, the running time will be much less than $\mathcal{O}(|V|^2)$ and close to $\mathcal{O}(|V| \log |V|)$ depending on the specific f.

While the acceleration that the lazy evaluation trick brings to the algorithm is good, it creates significant memory cost. The peak memory taken by the collection of priority queues is proportional to $m|V| = |V|^2/k$ (assuming the priority queue implementation has linear memory cost), which could be extreme for a large number of blocks. To mitigate the computation and memory efficiency issue of Algorithm 8, and also generate mini-batches with better order consistency (which is critical if the mini-batch size k is small relative to $|V|$

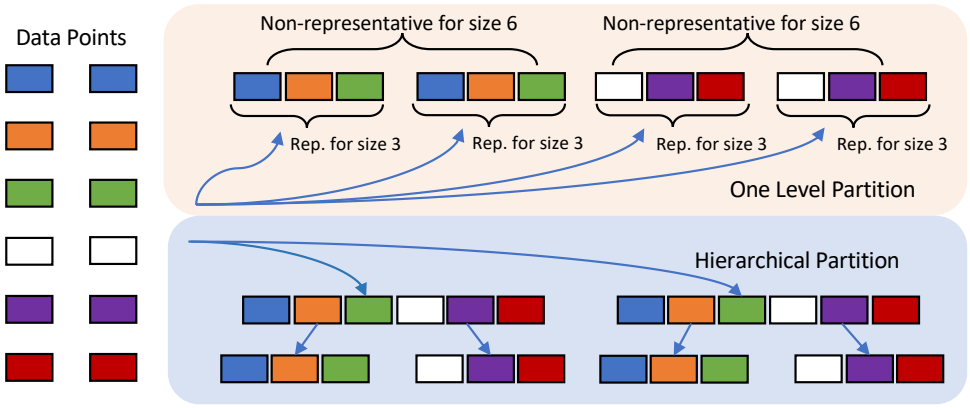


Figure 4.1: Simple example about how hierarchical partitioning can enforce better order consistency. Suppose we partition a dataset of 12 samples into mini-batches of size 3, and representativeness can be defined as the number of samples with distinct colors. In the upper part, though every mini-batch is representative for its size, the combination of either the first two or the last two mini-batches becomes non-representative for a combined block of size 6 (3 different colors for a set of size 6). On the other hand, the lower part shows the case where we first partition the data into blocks of size 6, and then further partition into mini-batches of size 3, which in this example enforces the representativeness of the combination of blocks.

and every mini-batch cannot represent the entire dataset), we propose a hierarchical robust partitioning framework, which runs Algorithm 8 with various block size constraint k's on different hierarchical levels.

We also note that lazier-than-lazy greedy (LTLG) [82] can also be applied to line 5 of Algorithm 8 to speed up the greedy step. Depending on the sample size of LTLG method, we can trade-off the performance of the algorithm with the memory and computational cost. If efficiency is the priority, we can apply LTLG on top of our method to achieve a better speed-up at the cost of some performance.

4.2.1 Hierarchical Robust Submodular Partition

In Chapter 3, we describe a hierarchical partitioning algorithm (Algorithm 10) that has potentially reduced computation and memory costs. Essentially, instead of having one block size constraint k and partitioning only once, we have a hierarchy of constraints $k_1 > k_2 > \ldots > k_r$, and ideally $k_i \bmod k_{i+1} = 0$. We start by partitioning the ground set into blocks of size k_1, and for every block we get from running Algorithm 8, we further partition each block into smaller blocks with a block size constraint k_2 and so on. In the end, k_r is the mini-batch size, so we get representative mini-batches.

By using Algorithm 10, we significantly reduce the memory cost of applying the lazy evaluation greedy trick. For iteration i of Algorithm 10, the peak memory cost is proportional to $m_i k_{i-1}$ (note $k_0 = |V|$) (assuming the memory cost of priority queues increases linearly), and the overall peak memory cost is $\max_{i=1:r} m_i k_{i-1}$. It is easy to see that $\forall i = 1 : r, m_i k_{i-1} \leq m|V|$, which is the peak memory cost of Algorithm 8 (note $m|V| = m_1 m_2 ... m_r k_0$). In fact, if we have $r = 2$ and $k_1 = |V|/2$, the memory cost is halved, and the more layers we have in the hierarchy, the less the overall memory cost becomes.

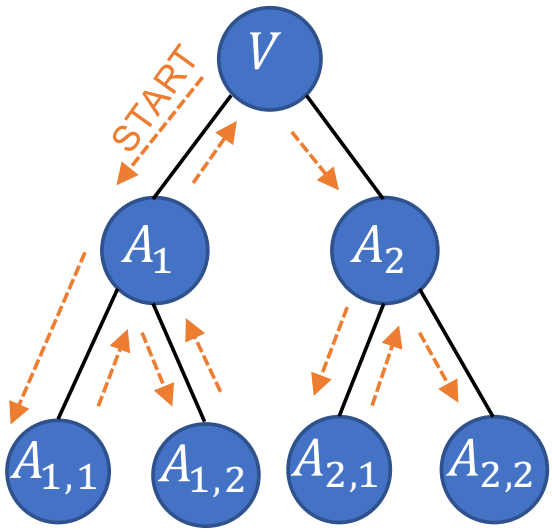


Figure 4.2: An example partition hierarchy. Leaves visited by any depth-first-search order is consistent with the partial ordering.

In addition to the memory efficiency with the lazy evaluation trick, Algorithm 10 also enforces groups of mini-batches in the hierarchy structure to be representative. Essentially, as we have a hierarchy of block size constraints $k_1, ..., k_r$, not only the final (lowest) level mini-batches are representative for their size, but the combination of multiple mini-batches is also representative for their combined size based on the choice of k's (see Figure 4.1 for a simple illustration). The original partitioning algorithm is a special case of the hierarchical partitioning with one level in the structure of hierarchy. As mentioned above, Algorithm 8 only applies to the case where the mini-batch is large enough to represent the dataset, and Algorithm 10 applies to general mini-batch sizes, while also making the original algorithm more efficient. Also, note that the generated hierarchy structure defines a partial ordering of the data samples, and any depth-first-search traversal of the hierarchy structure is consistent with the partial ordering (see Figure 4.2). We can also continue to use the decreasing submodular evaluation ordering as well (Algorithm 8 line 9). In terms of the approximation guarantee, recall that we prov the following bound under some mild assumption about the partitioning process:

Theorem 10 (**Min-Block Hierarchy**). *If we have $\tau \geq 2$ as defined in Def. 5 for every call to Algorithm 8 from Algorithm 10, then we achieve an approximation ratio of $(\frac{\tau-1}{2\tau-1})^r \frac{k_r}{|V|}$.*

τ as defined in Def. 5 indicates whether the blocks are filled in a balanced manner in Algorithm 8. $\tau \geq 2$ means that when the worst block (i.e., the block with minimal evaluation) has size k', and all other blocks have higher evaluations, the smallest block (in terms of size) has at least two elements, or in other words, the greedy steps do not generate drastically imbalanced blocks. In practice, for the real datasets in our experiments, τ always has large values, so the extra factor we get from Theorem 10 compared to Theorem 9 is close to 2^{-r}, and r is the number of layers in the hierarchy structure, which is at most $\log |V|$ and typically quite small in practice (3 to 5 in our experiments). This means Algorithm 10 will perform well.

The $\{k_i\}_{i=1}^r$ values are hyper-parameters for our method. Suppose the desired mini-batch size is large enough to represent the entire dataset, then the choices of $\{k_i\}_{i=1}^r$ essentially should follow the memory capacity of the hardware. However, for real-life problems, the

data may be very complex, and the mini-batch size is restricted by the GPU's memory size, and thus quite small compared to the dataset size. For example, in ImageNet, there are over one million training data points in 1000 classes, while the most widely adopted batch sizes on a single GPU card range from 128 to 512. The batch size of 128 can hardly be fully representative, as there are 1000 distinct classes. Therefore, Algorithm 10 is indeed required for partitioning such datasets as a poor order could happen similar to the case shown in Figure 4.1. Thus, one guideline for setting $\{k_i\}_{i=1}^r$ in addition to the efficiency restriction is to have $\{k_i\}_{i=1}^r$ start with multiple times of the number of classes, and gradually drill down to the desired mini-batch size. Algorithm 10 generates mini-batches of better quality than the ordinary robust partitioning case when we encounter datasets with more number of classes and larger data point size.

4.3 Experiments

We evaluate our hierarchical robust partitioning algorithm for generating mini-batch sequences for the CIFAR-100 [69] and ImageNet(ILSVRC12) [102] datasets. We show that our deterministically generated sequences consistently outperform randomly generated sequences, a widely-adopted approach for training deep neural networks. The initialization of the two datasets are fixed by random seed. For CIFAR-100, we use the PyTorch toolkit [96], and for ImageNet, we use MXNET [21].

Choice of Submodular Function

Our objective defined in Eq. (4.1) is a general framework for generating deterministic mini-batch sequences as the class of submodular function f can be extremely flexible and expressive for describing the representativeness of a set of data points. For this experiment, we use a nearest-neighbor submodular function, which is a variation of the facility location function as our choice of f for Eq. (4.1) for supervised classification tasks, i.e.,

$$f_{NN}(S) = \sum_{v \in V} \max_{v' \in S} \text{sim}(v, v'), \tag{4.2}$$

where $\text{sim}(\cdot)$ is defined over a pair of data points, and outputs the similarity of the two points. f_{NN} is a special case of the facility location function, with similarities between data points with different class labels being zeros (a sparse similarity graph), i.e. $y(v) \neq y(v') \rightarrow \text{sim}(v, v') = 0$, where $y(v)$ denotes the label of data point v. Variations of the facility location function have been successfully applied to various data selection/summarization tasks [138, 137, 126].

f_{NN} naturally captures the maximum likelihood estimates over the given data set for a nearest-neighbor classifier [140]. Under mild assumptions, maximizing f_{NN} is equivalent to finding the optimal subset of data points to form a nearest-neighbor classifier. Ideally, given a specific model (e.g., a nearest-neighbor classifier or a deep neural network), we should design f based on the objective of the model, so the exact quantization of representativeness defined by f is consistent with the given model. We choose to use f_{NN} because the nearest-neighbor classifier is quite generic, and f_{NN} is efficient to compute even for large datasets. We calculate the similarities between pairs of data points using: $\text{sim}(v_1, v_2) = e^{\frac{||x(v_1)-x(v_2)||_2}{-\sigma}}$, where $x(v)$ represents the vector representation or features of data point v, and σ is a chosen parameter, which works as a normalization factor. In practice, $x(v)$ can be the original data representation (e.g. raw pixel values for images), the final layer output of a deep neural network classifier, or the bottleneck layer of an autoencoder. While σ can be tuned, in principle we set it to $mean_{i,j}||x(v_i) - x(v_j)||_2$ for all the i, j pairs in the f_{NN} similarity graph.

CIFAR-100 Dataset

CIFAR-100 dataset is an image classification dataset with 50,000 training images and 10,000 testing images, with each image of dimension $(3, 32, 32) = 3072$. The dataset has 100 classes, and all the classes have the same number of training/testing data points. We randomly select 100 samples from each class in the training set as the validation set. We train a convolutional autoencoder network to extract more compact representations of the raw pixel data. The autoencoder network in use has 48 layers with a 128-dimensional bottleneck layer and was trained on the large TinyImage data set [125]. The network employs residual blocks [41].

Furthermore, the training was performed using ADAM [62] with batch normalization and rectified linear units. The output of the bottleneck layer act as the 128-dimensional feature representation of the data, which is then used to compute the similarities between data points for f_{NN}.

The deterministic data sequence is generated through a multi-step mechanism. First, we generate a sparse similarity graph using the similarity metric discussed in Sec 4.3 (containing $400^2 \times 100 = 16M$ entries) that can be used to instantiate f_{NN}. Next, we apply Algorithm 10 with f_{NN} as the submodular function, and block size constraints $k_1 = 1024$, $k_2 = 512$ and $k_3 = 128$, where k_3 is equal to the mini-batch size used for training the deep neural network. We also note that for the data points that are not selected by Algorithm 10 (40000 mod 1024 = 64), we form a final batch (padded depending on the implementation of the toolkit) and append to the end of our mini-batch sequence.

For a given sequence of mini-batches, we train a Wide Res-net [147] having structure WRN-28-8 (using the same terminology as [147]). The network was trained using the NAG [121] optimization method, an initial learning rate of 0.1 and a linearly decaying learning rate schedule. The data was augmented using random flipping of images. We note that the training hyper-parameters are determined by manually tuning on the validation set accuracy while training on a randomly generated sequence (same for ImageNet dataset described below). We compare the validation set accuracy, and test set accuracy of our deterministically generated sequence to 10 randomly generated sequences. Figure 4.3 demonstrates that our method outperforms the random generated sequences, and can also do better than the best over 10 random runs.

ImageNet Dataset

The ImageNet classification dataset contains roughly 1.2 million training images and 1000 classes. The number of data points for each class is almost balanced, and each class has around 1k data points. The validation set consists of around 50k samples, and there is no standard test set. We train a deep neural net classifier and retrieve the final layer out-

put (before softmax) as the feature representations for data points. The DNN classifier is a Residual Network [43] with 18 layers (Resnet-18), and the feature representation has 1000 dimensions. Similar to the CIFAR-100 dataset case, we use the extracted 1000 dimensional features to generate the sparse similarity graph. Then we utilize Algorithm 10 with $k_1 = 32768$, $k_2 = 16384$, $k_3 = 8192$, $k_4 = 4096$, $k_5 = 128$ to generate the deterministic sequence of mini-batches. For the non-selected data points (3215 data samples, since the total number of samples does not divide k_1), we form a new ground set and run Algorithm 8 to partition them into mini-batches of size 128 and append them to the end of the sequence.

We compare the performance of different sequences using the same model used to generate features for data points, i.e., the Resnet-18 network structure. For both cases, we use again the NAG [121] optimization method with an initial learning rate 0.1, exponentially decaying learning rate schedule, and data augmentation of random flipping and random cropping. We note that for training the feature extraction model, we use a deterministic sequence generated using the L2 distance of the raw data as the distance measurement for constructing the similarity graph.

As shown in Figure 4.4, for most cases, our deterministic sequence yields better results than even the best among the 10 randomly generated sequences, significantly beating the mean and the worst random, which are both likely to happen when training deep models.

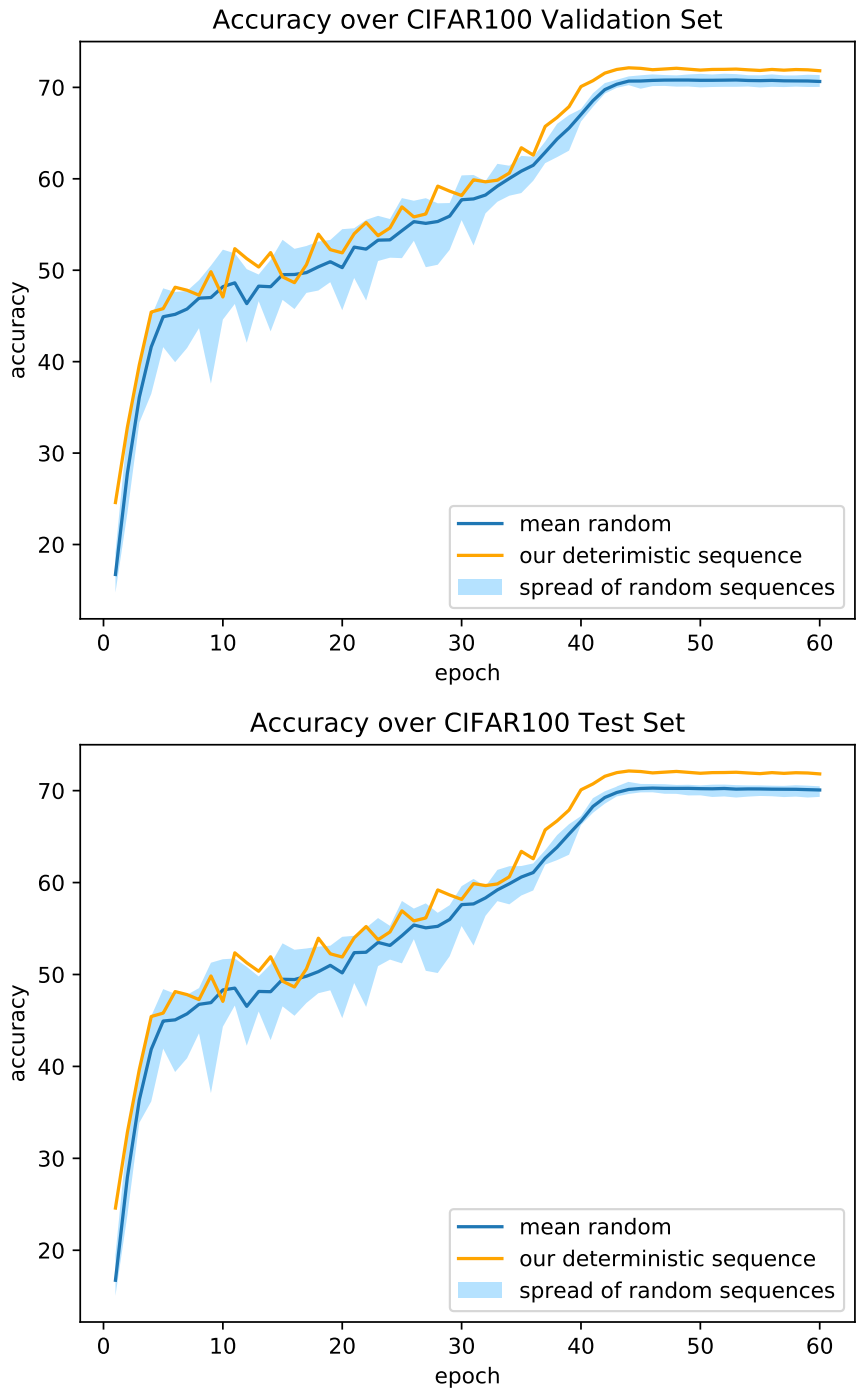


Figure 4.3: (top) Accuracy over the validation set for our sequence along with the performance spread for the random sequences. (bottom) Accuracy over the test set for our sequence along with the performance spread for the random sequences.

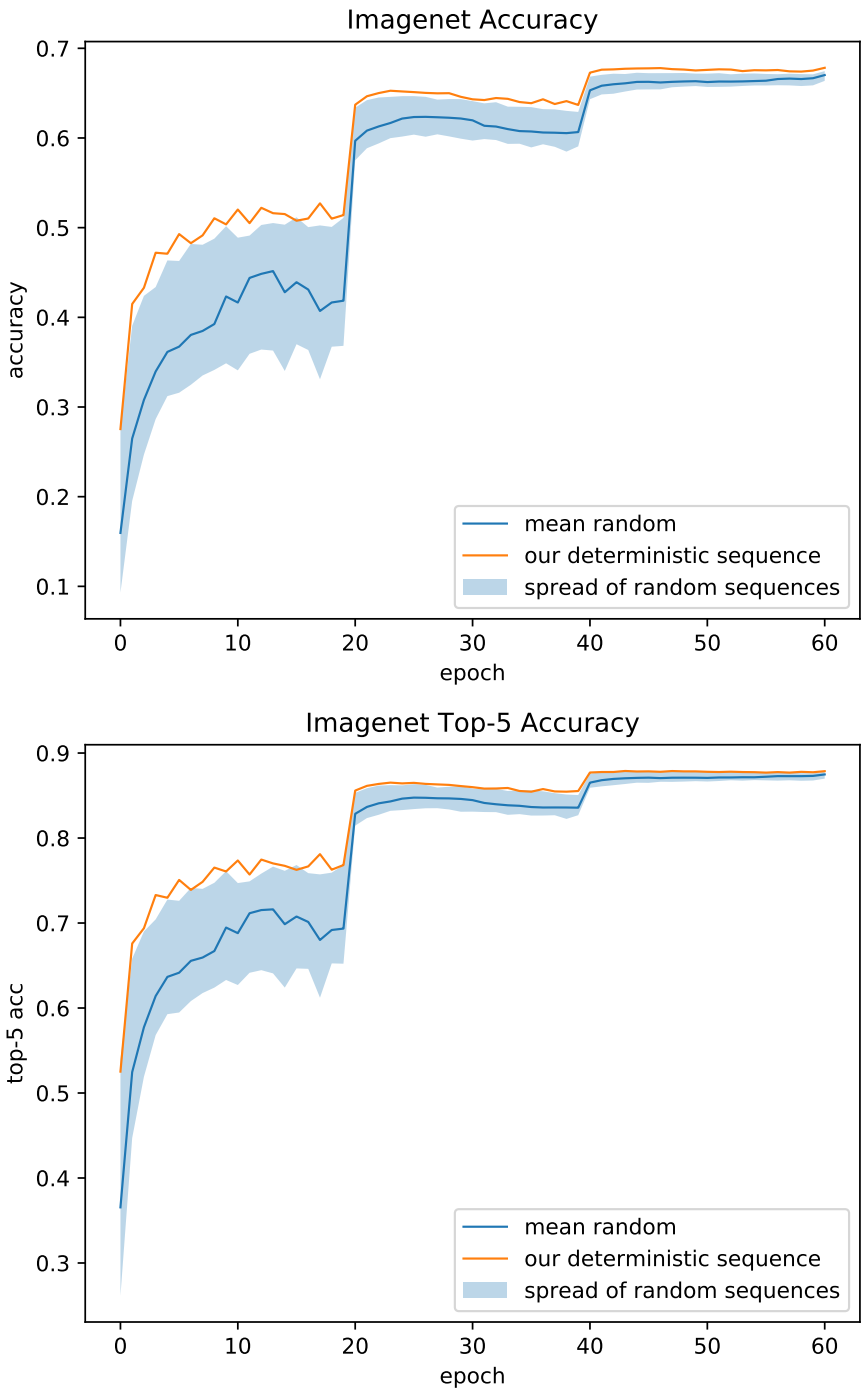


Figure 4.4: (top) Accuracy over the validation set for our sequence along with the performance spread for the random sequences. (bottom) Top-5 accuracy over the validation set for our sequence along with the performances spread for the random sequences.

Part II

LINEAR MODELS OF RELU DEEP NETWORKS

In the second part of this thesis, we focus on investigating a deep ReLU network by decomposing it as a collection of linear models for different data points. We first study the spectrum, i.e., the singular value distribution of the Extended Data Jacobian Matrix (EDJM) of a deep ReLU network, where each row of the EDJM is a linear model for one data point. The spectrum of the EDJM is found to be highly correlated with the complexity of the learned functions after studying the effect of dropout, ensembles, and model distillation using EDJM. Next, we show an efficient regularization method Jumpout, an improved version of dropout, based on linear models of ReLU networks. We propose three novel modifications of dropout: 1) change the dropout rate to make the regularization locally smooth; 2) normalize the dropout rate by the proportion of active neurons of the ReLU layer; 3) modify the rescaling factor to make dropout synergize better with batchnorm layers. Finally, we aim to explain the network behavior based on the linear model for every data point, particularly based on the bias term of the linear model. We propose a backpropagation-type algorithm "bias backpropagation (BBp)" that starts at the output layer and iteratively attributes the bias of each layer to its input nodes as well as combining the resulting bias term of the previous layer. In experiments, we show that BBp can generate complementary and highly interpretable explanations.

Chapter 5
INTRODUCTION

5.1 Background

Deep neural networks (DNNs) have produced good results for many challenging problems in computer vision, natural language processing, and speech processing. DNNs are a highly expressive trainable class of non-linear functions, utilizing multi-layer architectures and a rich set of possible hidden non-linearities. A simple and widely adopted non-linearity function is the rectified linear unit (ReLU), which keeps the positive part of the input while setting the negative part to be zero. More generally, ReLU is a specific form of piecewise linear activation functions. Because of the piecewise linear property, the overall network is a piecewise linear model over the input data space.

Since a DNN is piecewise linear for the ReLU activation function, the model is essentially a linear model in a local region. Precisely, for every data point, we can modify the weight matrices of the DNN according to the activation patterns of the ReLU non-linearities such that by setting the rows of the weight matrices corresponding to the non-active ReLU neurons to be all zeros, we can remove the ReLU activation functions and get a multiplication of multiple modified weight matrices, resulting in a linear model for that data point. As the model is linear, the weight of the linear model is equal to the gradient of the network output with respect to the input data point. Note that the DNN behaves as different linear models for different data points, so overall it is a piecewise linear model.

The natural decomposition of the ReLU (or more generally DNNs with piecewise linear activation functions) as a linear model for every data point allows us to investigate and analyze the network's behaviors based on such linear models. For the following, based on such intuition, we present three works to better analyze, improve and interpret DNNs with

ReLU activation functions: 1) we collect the linear models for different training data points into a matrix and analyze its spectrum (i.e., singular value distribution); 2) we improve the dropout layer by enforcing smooth predictions of nearby linear models; 3) we develop an attribution/interpretation method specific for the bias term in the linear model for every data point.

5.2 Spectra of Extended Data Jacobian Matrix

Deep Learning has achieved significant success in the past few years on many challenging tasks, but there is still much work needed to understand why deep architectures are able to learn such effective representations. One proposal [95, 87] argues that deep neural networks with rectified units are able to separate the input space into exponentially more linear response regions than shallow networks given the same number of computational units, thus enabling deep networks to learn highly complex and structured functions. Empirical results in [101] show that deep but thin networks can mimic the function learned by shallower (but still deep) and wide networks. Even for deep linear networks, [104] found that they exhibit non-linear training dynamics similar to non-linear deep neural networks. Although deep linear networks are equivalent to shallow linear networks in representational power, their training dynamics are rich in mathematical structure that alter key aspects of the network such as training time, and the effects of pre-training, random initialization, and generalization. Better understanding of the trajectory of the functions learned by different architectures during and at the end of training is important for understanding why deep learning is so effective, how different architectures affect what is learned, and for developing networks with better regularization and generalization abilities. For example, in [93], understanding and enforcing scale invariance in ReLU networks led to developing a more resilient update rule than standard SGD for networks with unbalanced weights.

We started this work motivated by trying to understand what qualities deep networks possess that shallow networks do not, and also how convolutional neural networks differ from feed-forward ones. Recently, model compression and distillation [3, 47] show that it

is possible to train compact models to approximate the functions learned by more complex models. Empirically, on some datasets, shallow networks can be improved dramatically to the point of even matching the best deep network by learning from the soft labels generated by a large and deep model or an ensemble of such models. Such improvement may result from the extra information provided by the soft labels which preserves the relative confidence of different outputs learned by the bigger model. The distillation results posed yet another interesting question: in what way does the more complex, "teacher", network alter the function learned by the less complex, "student", network compared to a similar network learning without the benefit of a teacher?

Conditioned on a set of data points that define the manifold of interest, we propose the Extended Data Jacobian Matrix (EDJM) as an analysis tool for neural networks. By studying the spectrum of EDJM, which we believe is highly correlated with the complexity of the functions learned by networks, we can compare networks with different depths, architectures, and training techniques in a unified way. First, we introduce the Extended Data Jacobian Matrix for feed-forward and convolution neural networks. Then we show the characteristics of the spectrum of the EDJM for the best performing networks over multiple datasets. We also show that different methods such as model compression and dropout increases the same measure of the spectrum of the EDJM. Finally, motivated by these observations, we propose a regularization technique for improving performance of deep neural networks.

5.3 Jumpout: Improved Dropout for ReLU Network

Dropout is a powerful and simple regularization technique for training deep neural networks. However, it has several drawbacks. Firstly, dropout rates, constituting extra hyper-parameters at each layer, need to be tuned to get optimal performance. Too high a dropout rate can slow the convergence rate of the model, and often hurt final performance. Too low a rate yields few or no improvements on generalization performance. Another deficiency of dropout lies in its incompatibility with batch normalization (BN) [50] (more empirical evidence of this is shown in Section 7.2.3). As dropout randomly shuts down activated neurons,

it needs to rescale the undropped neurons to match the original overall activation gain of the layer. Unfortunately, such rescaling breaks the consistency of the normalization parameters required between training and test phases[1] and may cause poor behavior when used with BN.

We propose three simple modifications to dropout in order to overcome the aforementioned drawbacks, leading to an improved method we call "jumpout." Our approach is motivated by three observations about how dropout results in improved generalization performance for DNNs with rectified linear unit (ReLU) activations, which covers a frequently used class of DNNs.

Firstly, we note that any DNN with ReLU is a piecewise linear function which applies different linear models to data points from different polyhedra defined by the ReLU activation patterns. Based on this observation, applying dropout to a training sample randomly changes its ReLU activation patterns and hence the underlying polyhedral structure and corresponding linear models. This means that each linear model is trained not only to produce correct predictions for data points in its associated polyhedron, but also is trained to work for data points in nearby polyhedra; what precisely "nearby" means depends on the dropout rate used. This partially explains why dropout improves generalization performance. The problem, however, is that with a fixed dropout rate, say p, and on a layer with n units, the typical number of units dropped out is np as that is the mode of a Binomial distribution with parameter p. It is relatively rare that either very few (closer to zero) or very many (closer to n) units are dropped out. Thus, with high probability, each linear model is smoothed to work on data points in polyhedra at a typical distance np away. The probability of smoothing over closer distances is much smaller, thus failing to achieve the goal of local smoothness.

In jumpout, by contrast, p rather than being fixed is itself a random variable; we sample p from a distribution that is monotone decreasing (e.g., a truncated half-Gaussian). This

[1]Dropout usually happens only during the training but not the test phase, since using it for testing requires averaging the results of multiple dropout inferences on each training sample, which is costly and may introduce greater prediction variance.

achieves the property that Pr(i units dropping out) $\geq$ Pr(i+1 units dropping out) for all $i \in \{1, 2, \ldots, n\}$. That is, a smaller dropout rate has a higher probability of being chosen. Hence, the probability of smoothing polyhedra to other points decreases as the points move farther away. Secondly, we notice that in dropout, the fraction of activated neurons in different layers, for different samples and different training stages, can be different. Although we are using the same dropout rate, since dropping out neurons that are already quiescent by ReLU changes nothing, the effective dropout rate, i.e., the fraction of the activated neurons that are dropped, can vary significantly. In jumpout, we adaptively normalize the dropout rate for each layer and each training sample/batch, so the effective neural-deactivation rate applied to the activated neurons are consistent over different layers and different samples as training proceeds. Lastly, we address the incompatibility problem between dropout and BN by rescaling the outputs of jumpout in order to keep the variance unchanged after the process of neural deactivation. Therefore, the BN layers learned in the training phase can be directly applied in the test phase without an inconsistency, and we can reap the benefits of both dropout and BN when training a DNN. In our experiments on a broad range of benchmark datasets including CIFAR10, CIFAR100, Fashion-MNIST, SVHN, STL10 and ImageNet-1k, jumpout shows almost the same memory and computation costs as the original dropout, but significantly and consistently outperforms dropout.

5.4 Bias Attribution for Deep Network Explanation

Deep learning models are usually designed using fairly high-level architectural decisions, leading to a final model that is often seen as a difficult to interpret black box. This restricts the reliability and usability of DNNs especially in mission-critical applications where a good understanding of the model's behavior is necessary.

The gradient is a useful starting point for understanding and generating explanations for the behavior of a complex DNN. Having the same dimension as the input data, the gradient can reflect the contribution to the DNN output of each input dimension. Not only does the gradient yield attribution information for every data point, but also it helps us

understand other aspects of DNNs, such as the highly celebrated adversarial examples and defense methods against such attacks [124]. When a model is linear, the gradient recovers the weight vector. Since a linear model locally approximates any sufficiently smooth non-linear model, the gradient can also be seen as the weight vector of that local linear model for a given DNN at a given data point. For a piecewise linear DNN (e.g., a DNN with activation functions such as ReLU, LeakyReLU, PReLU, and hard tanh) the gradient is exactly the weights of the local linear model.

Although the gradient of a DNN has been shown to be helpful in understanding the behavior of a DNN, the other part of the locally linear model, i.e., the bias term, to the best of our knowledge, has not been studied explicitly and is often overlooked. We unveil the information embedded in the bias term by developing a general bias attribution framework that distributes the bias scalar to every dimension of the input data. We propose a backpropagation-type algorithm called "bias backpropagation (BBp)" to send and compute the bias attribution from the output and higher-layer nodes to lower-layer nodes and eventually to the input features, in a layer-by-layer manner. Specifically, BBp utilizes a recursive rule to assign the bias attribution on each node of layer ℓ to all the nodes on layer $\ell - 1$, while the bias attribution on each node of layer $\ell - 1$ is composed of the attribution sent from the layer below and the bias term incurred in layer $\ell - 1$. The sum of the attributions over all input dimensions produced by BBp exactly recovers the bias term in the local linear model representation of the DNN at the given input point. In experiments, we visualize the bias attribution results as images on a DNN trained for image classification. We show that bias attribution can highlight essential features that are complementary with what the gradient-alone attribution methods favor.

5.5 Road Map of Part II

In Chapter 6, we introduce the decomposition of a ReLU network as a collection of linear models for different data points. We analyze the spectra of the EDJM under different settings: networks with different depths, dropout, knowledge distillation and etc.

In Chapter 7, we propose an improved version of Dropout called Jumpout specifically for ReLU networks with three modifications that draw intuition from the linear models of ReLU networks.

In Chapter 8, we propose a backpropagation-type algorithm "bias back-propagation (BBp)" to explain the network's behavior based on the bias terms of the linear models for different data points.

Chapter 6 was originally presented in the following paper:

- S Wang, A Mohamed, R Caruana, J Bilmes, M Phlipose, M Richardson, K Geras, G Urban, O Aslan. *Analysis of Deep Neural Networks with Extended Data Jacobian Matrix*, In ICML 2016.

Chapter 7 was originally presented in the following paper:

- S Wang*, T Zhou*, J Bilmes. *Jumpout: Improved Dropout for Deep Neural Networks with ReLUs*, In ICML 2019.

Chapter 8 was originally presented in the following paper:

- S Wang*, T Zhou*, J Bilmes. *Bias also matters: Bias attribution for Deep Neural Network Explanation*, In ICML 2019.

Chapter 6

SPECTRA OF EXTENDED DATA JACOBIAN MATRIX

Deep Learning has achieved significant success in the past few years on many challenging tasks [70, 46, 113, 8, 83, 111, 122] , but there is still much work needed to understand why deep architectures are able to learn such effective representations.

One proposal [95, 87] argues that deep neural networks with rectified units are able to separate the input space into exponentially more linear response regions than shallow networks given the same number of computational units, thus enabling deep networks to learn highly complex and structured functions. Empirical results in [101] show that deep but thin networks can mimic the function learned by shallower (but still deep) and wide networks. Even for deep linear networks, [104] found that they exhibit non-linear training dynamics similar to non-linear deep neural networks. Although deep linear networks are equivalent to shallow linear networks in representational power, their training dynamics are rich in mathematical structure that alter key aspects of the network such as training time, and the effects of pre-training, random initialization, and generalization. Better understanding of the trajectory of the functions learned by different architectures during and at the end of training is important for understanding why deep learning is so effective, how different architectures affect what is learned, and for developing networks with better regularization and generalization abilities. For example, in [93], understanding and enforcing scale invariance in ReLU networks led to developing a more resilient update rule than standard SGD for networks with unbalanced weights.

We started this work motivated by trying to understand what qualities deep networks possess that shallow networks do not, and also how convolutional neural networks differ from feed-forward ones. Recently, model compression and distillation [3, 47] show that it is possi-

ble to train compact models to approximate the functions learned by more complex models. Empirically, on some datasets, shallow networks can be improved dramatically to the point of even matching the best deep network by learning from the soft labels generated by a large and deep model or an ensemble of such models. Such improvement may result from the extra information provided by the soft labels which preserves the relative confidence of different outputs learned by the bigger model. The distillation results posed yet another interesting question: in what way does the more complex, "teacher", network alter the function learned by the less complex, "student", network compared to a similar network learning without the benefit of a teacher?

Conditioned on a set of data points that define the manifold of interest, we propose the Extended Data Jacobian Matrix (EDJM) as an analysis tool for neural networks. By studying the spectrum of EDJM, which we believe is highly correlated with the complexity of the functions learned by networks, we can compare networks with different depths, architectures, and training techniques in a unified way. First, we introduce the Extended Data Jacobian Matrix for feed-forward and convolution neural networks. Then we show the characteristics of the spectrum of the EDJM for the best performing networks over multiple datasets. We also show in section 6.3, that different methods such as model compression and dropout increases the same measure of the spectrum of the EDJM. Finally, motivated by these observations, we propose a regularization technique for improving performance of deep neural networks.

6.1 Data Jacobian Matrix

Given a deep neural network consisting of m *linear* layers, the entire network is a series of matrix multiplies that can be represented merely by a single matrix:

$$W_{net} = W_m W_{m-1} \dots W_1 \tag{6.1}$$

Suppose we are given a data set D, and $|D| = n$, where each data point $(x, y) \in D$ consists of input features x of d_{in} dimensions, and an output label y of d_{out} dimensions. Using the linear network described above, we have, for (x_i, y_i):

$$\hat{y}_i = W_m W_{m-1} \dots W_1 x_i, \quad \text{or} \quad \hat{y}_i = W_{net} x_i, \tag{6.2}$$

where $\hat{y}_i$ is the predicted label of x_i using the network.

Now we introduce non-linearity into the linear network. Without loss of generality, we use Rectified Linear Units (ReLU), which sets negative entries in a vector to 0 while preserving the positive entries (We will show how to generalize to other types of activation functions later):

$$\phi_{ReLU}(z) = \max(0, z) \tag{6.3}$$

If we apply ReLU to all the linear layers except for the last layer, we have the following network:

$$\hat{y}_i = W_m \phi_{ReLU}(W_{m-1} \phi_{ReLU}(\dots \phi_{ReLU}(W_1 x_i))) \tag{6.4}$$

6.1.1 ReLU Network as a Collection of Linear Systems

Suppose for input x_i, the output at layer l is h_l^i, then $h_l^i = \phi_{ReLU}(W_l h_{l-1}^i)$. If $(W_l h_{l-1}^i)[k] < 0$, then $h_l^i[k] = 0$ according to the ReLU function, which is equivalent to setting the k th row of W_l to be all 0, while if $(W_l h_{l-1}^i)[k] \geq 0$, the row is left exactly the same. Therefore, we can eliminate the non-linear ReLU functions by modifying certain rows of weight matrices to 0 while keeping the rest, and we end up with a linear system for data point x_i:

$$\hat{y}_i = W_m \hat{W}_{m-1}^i \hat{W}_{m-2}^i \dots \hat{W}_1^i x_i \tag{6.5}$$

$$\hat{y}_i = W_{net}^i x_i \tag{6.6}$$

where:

$$\hat{W}_l^i[a, b] = \begin{cases} W_l[a, b] & \text{if } h_l^i[a] > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{6.7}$$

Note that W_{net}^i is x_i specific, as the modification that transforms W_l into $\hat{W}_l^i$ depends on the values of h_l^i, and therefore is based on the input features x_i. For each individual data

point in a finite data set, we can construct a linear model W^i_{net} of dimensions $d_{out} \times d_{in}$ in such manner, and that generates the same output as the original neural network consisting of arbitrary number of layers and hidden units.

Clearly, for different data points x_i and x_j, different linear systems will most likely be constructed. However, for an x_j that is close enough to x_i, the output of the first layer h^i_1 and h^j_1 could have the same patterns of ReLU activations In other words, $(W_1 x_i)[k] < 0 \iff (W_1 x_j)[k] < 0 \ \forall k$, so in this case, the modified weight matrices are the same for both data points. If such pattern is carried over multiple layers, the constructed linear systems could be the same. Therefore, for input data points in a small region around x_i, the constructed linear system for x_i remains unchanged, so that the ReLU network is piece-wise linear.

6.1.2 Data Jacobian Matrix

For dataset $D = \{(x_i, y_i)\}$, where x is of dimension d_{in} and y is of dimension d_{out}, and a deep neural network $\hat{y}_i = f_m(\phi_{m-1}(f_{m-1}(\phi_{m-2}(\dots \phi_1(f_1(x_i))))))$, where f is a linear operation, and ϕ is a differentiable non-linear function, we denote the *Data Jacobian Matrix* for input features x_i as:

$$DJM_\theta(x_i) = \frac{\partial \hat{y}_i}{\partial x_i} = \frac{\partial \hat{y}_i}{\partial h^i_{m-1}} \frac{\partial h^i_{m-1}}{\partial h^i_{m-2}} \cdots \frac{\partial h^i_1}{\partial x_i} \tag{6.8}$$

$$= \frac{\partial f_m(h^i_{m-1})}{\partial h^i_{m-1}} \frac{\partial h^i_{m-1}}{\partial f_{m-1}(h^i_{m-2})} \cdots \frac{\partial f_1(x_i)}{\partial x_i}. \tag{6.9}$$

The subscript θ in the DJM denotes that depends on the neural network parameters θ. For ReLU feed-forward network in specific, we have:

$$DJM_\theta(x_i) = W_m \frac{\partial h^i_{m-1}}{\partial W_{m-1} h^i_{m-2}} W_{m-1} \frac{\partial h^i_{m-2}}{\partial W_{m-2} h^i_{m-3}} \dots W_1$$

In addition, under ReLU non-linear function, we have:

$$\frac{\partial h^i_{l+1}[a]}{\partial (W_{l+1} h^i_l)[b]} = \begin{cases} 1 \text{ if } a = b \text{ and } (W_{l+1} h^i_l)[b] \geq 0 \\ 0 \text{ otherwise} \end{cases}.$$

It is easy to see that:

$$DJM_\theta(x_i) = W_m \hat{W}^i_{m-1} \hat{W}^i_{m-2} \dots \hat{W}^i_1 = W^i_{net} \tag{6.10}$$

$$\hat{y}_i = DJM_\theta(x_i) x_i, \tag{6.11}$$

For ReLU feed-forward network, the Data Jacobian Matrix for x_i is equivalent to the per-data point linear system for x_i (Eq 6.11). Such equivalence also holds for ReLU convolutional networks with max/average pooling layers. Convolution, as a linear transformation, can be written in the form of matrix multiplication, and thus no different from the feed-forward layers when calculating the gradients. Average pooling is a special case of convolution, which can be viewed as a matrix multiplication as well. Max pooling outputs the highest value out of a certain receptive field, which is equivalent to removing certain rows of the weight matrix based on the input values. Similar to the ReLU function, when calculating the gradients, elements in the removed rows receive 0 gradients and therefore the equivalence holds.

Conceptually, the Data Jacobian Matrix is the gradient of the outputs of the neural network with respect to the inputs. Suppose the neural network is applied to a classification task, ReLU networks are characterized with piece-wise linear classification boundaries, which collide with the gradients. For non-linear functions other than ReLU (e.g. sigmoid, and tangent), the gradients are linear approximations to the classification boundaries.

6.2 Extended Data Jacobian Matrix

For simplicity, suppose $d_{out} = 1$ (i.e. the task is binary classification or a single value regression), for data point x_i, $DJM_\theta(x_i)$ is a single vector. Stacking such vectors across different data points in X with $|X| = n$, we get a matrix of dimension $n \times d_{in}$. We denote such a matrix as the *Extended Data Jacobian Matrix* (EDJM). In the general case, for every dimension of the output, we can construct such Extended Data Jacobian Matrix, so that we have d_{out} EDJMs all of which are of dimension $n \times d_{in}$.

The Extended Data Jacobian Matrix is a collection of linear systems across data points, allowing us to study and compare various neural networks conditioned on certain set of data.

For ReLU networks in particular, EDJM is equivalent to the neural network for the input dataset. For the rest of the chapter, we will focus on ReLU networks for the equivalence we get, yet all the analysis described in the following applies to non-ReLU networks as well.

An EDJM has fixed dimension $n \times d_{in}$ for a given data set regardless of the network structure, thus allowing us to compare different networks (e.g., deep vs. shallow, or convolutional vs. feed-forward) through the same lens.

The spectrum, or distribution of singular values, of EDJM reflects the principle components of the space spanned by different linear systems for different data points. We choose to use matrix factorization over each EDJM instead of tensor factorization on the tensor formed by d_{out} EDJMs for the following reasons: 1) tensor factorization in general is NP-hard and there is identifiability issue with tensor factorization; 2) the computational cost for tensor factorization in our experiment setting can be intractable; 3) matrix factorization of EDJMs can be used to investigate and compare different linear systems that map inputs to different output classes. A spectrum with one single component suggests that the linear system for certain output dimension is the same for all data points, making the neural network very simple, and indeed linear for that output dimension. The other extreme is when the spectrum of EDJM is uniform across all components meaning that the space spanned by the data points' linear systems is extremely complex and the underlying neural network behaves dramatically different for different data points. The distribution of singular values of the EDJM, therefore, naturally reflects the "complexity" of the function learned by the neural network. For the rest of this chapter, we will compare many neural networks with different architectures to demonstrate the connection to the spectrum of the EDJM.

6.3 Empirical Analysis with EDJMs

In this section, we will show that the EDJM spectrum correlates well with our natural notion of model complexity and more advanced training methods, as well as being predictive of model accuracy.

Experiments in this chapter are conducted on three different datasets: MNIST for hand-

written digit recognition, CIFAR-10 for image recognition, and TIMIT for phone recognition. MNIST consists of 60000 training data points, out of which we randomly extract 10000 data points as the validation set, and 10000 testing data points. Each data point is a single channel image of size $(1, 28, 28)$ and thus has 784 dimensional features. CIFAR-10 consists of 50000 training images and 10000 testing images. Similar to MNIST, we extract 10000 out of the training dataset as a validation set. Each image in the CIFAR-10 dataset is of size $(3, 32, 32)$, which is of dimension 3072. Both MNIST and CIFAR-10 has 10 output classes, so 10 Extended Data Jacobian Matrices can be constructed. For MNIST, the dimensions of the EDJM are $n \times 784$, and for CIFAR-10, the dimensions the EDJM are $n \times 3072$, where the value of n depends on the subset of data selected. For validation set, $n = 10000$ for both MNIST and CIFAR-10. The TIMIT corpus consists of a 462 speaker training set, a 50 speaker validation set, and a 24 speaker test set. 15 frames are grouped together as inputs where each frame contains 40 log mel filterbank coefficients plus energy along with their first and second temporal derivatives.

We use stochastic gradient descent with momentum for training all the following reported models. Learning rates gets halved if the performance does not improve over a succession of 5 epochs on the validation set. No regularization/batch normalization is applied if not specified. The reported models are all selected by grid search for best performance to cover a broad range for each parameter in order to ensure a fair comparison between models.

6.3.1 EDJM for Feed-forward Networks

We compare EDJM of neural nets with various depths (# layers $= 1, 2, 3, 4$) while we either fix the total number of parameters, or the total number of hidden units. Fixing the total number of parameters constrains the size and computational costs of the network. Moreover, for ReLU networks, fixing the number of hidden units restricts the upper bound on the number of different DJMs that can be generated. Specifically, for ReLU networks with H hidden units, at most 2^H different DJMs can be generated, as the activation/deactivation of ReLU on any hidden unit may contribute to construct a different linear system. Also note

that such upper bound is irrespective of the number of layers.

Plotted in Figure 6.1 is the normalized spectrum (normalized by the max singular value, so the curves start with 1.0 on the left) of EDJMs for different feed-forward networks conditioned on the validation set of MNIST and CIFAR-10, and averaged over 10 output dimensions. As we apply softmax to the output of the networks for classification tasks, which may result in significantly different scales for linear systems of different networks, we report the normalized spectrum. We observe same patterns for training set, validation set and testing set. We choose to present the validation set results as 1) a natural selection of the conditioning dataset for EDJMs is the data utilized during the training process; 2) the accuracy associated with each curve on the validation set is more informative than the training set; 3) analysis of the validation set results can be beneficial to tuning the hyper-parameters of training, which should never be done on the testing set. For all the following comparisons of spectra, we always report the validation set results.

From Figure 6.1, we observe a clear trend that EDJMs of deeper networks tend to have more dominating singular values. To quantify the plotted shape in one simple metric, we propose the following score:

$$score(S_\theta(X, j)) = \sum_{\sigma \in S_\theta(X,j), \sigma > \epsilon} \frac{\sigma}{\max(S_\theta(X, j))}, \tag{6.12}$$

where $S_\theta(X, j)$ denotes the singular values for EDJM conditioned on data input X and jth output. Conceptually, the score is the sum over normalized singular values excluding the small ones. From Figure 6.1 we observe that the singular values decrease at a super-linear rate, and therefore there are lot of small singular values, whose corresponding components are more likely to be noise for our analysis. On the other hand, the proposed score captures the relative power of the important components. Higher score suggests that there are more important components embedded in the given EDJM, or the underlying neural network represents a more complex set of linear systems. The value of ϵ is chosen to be 90 percentile of the singular values (we keep the top 10% singular values since the rest 90% values are

small and likely to be noise). The scores averaged over 10 output dimensions for different neural network structures are reported in Figure 6.1 (C) and (F), which coincide with the trend observed from the spectra.

For either fixed number of hidden units or fixed number of parameters, deeper networks have higher normalized singular values for the major components of the spectrum. The depth of the neural network acts like a prior on the spectrum of the EDJM: deeper neural networks are likely to be trained to generate a more complex set of linear systems conditioned on the dataset of interest, which is consistent with the common belief that more complex functions are learned with deep neural networks than shallow neural networks.

6.3.2 EDJM for Convolutional Networks

In Figure 6.2, we show a comparison between the best performing feed-forward network with convolutional networks that have comparable number of parameters. For both datasets, large gaps between the spectrum of convolutional and feed-forward networks are observed, suggesting that the learned functions for the convolution networks are tremendously more complex than the feed-forward ones.

Our contention is that the reason behind the large gaps is in the difference between the number of hidden units of the feed-forward and the convolutional networks. Suppose, for every filter, the output size is the same as the input size (i.e. we do not lose the boundary region when performing convolution). The total number of output units is therefore the number of filters $\times$ the input size. For CIFAR-10, suppose we have 64 filters as our first layer, we then end up with $32 * 32 * 64 = 65536$ hidden units. As stated above, the total number of hidden units puts a restriction on the upper-bound of the number of independent linear systems that can be constructed through the ReLU network. Therefore, convolution networks have a much higher upper-bound, which may result in more complex systems.

6.3.3 EDJM for Model Compression

So far, we have seen that EDJM scores vary with neural network architecture. In the rest of the chapter we will show that the scores also vary with the training technique utilized. Model compression or distillation [3, 47], focuses on training a compact model to approximate the function learned by a more complex model. Empirically, for certain datasets, a shallow (but not too small) network can be improved dramatically, even matching the best deep network, by learning from the soft labels generated by a large and complex model, or alternatively an ensemble of such models. Such improvements may result from the extra information provided by the soft labels, referred as "dark knowledge" in [45], which preserves the relative confidence of different outputs learned by the complex model.

We analyze the EDJM for models trained with the model compression framework to further illustrate EDJM's ability measure the relative complexity of models trained in very different ways, as well as to try to better understand the model compression process. Specifically, we compare the EDJM for five models on the CIFAR-10 dataset (the performance gap between the complex and simple models on MNIST without distillation is too small to be interesting). The five models are as follows:

1) Convolutional Teacher Model: A deep convolution network of structure: 64 filter 3 * 3 conv - 64 filter 3 * 3 conv - 2 * 2 max pooling - 128 filter 3 * 3 - 128 filter 3 * 3 - 2 * 2 max pooling - 256 filter 3 * 3 - 256 filter 3 * 3 - 256 filter 3 * 3 - 256 filter 3 * 3 - 2 * 2 max pooling - 2 * 1024 feed-forward layer, and trained with hard 0-1 labels. 2) Convolutional Student Model: A shallow convolution network of structure: 128 filter 5 * 5 - 2 * 2 max pooling - 800 linear layer - 1 * 5000 feed-forward layer, with soft labels provided by the convolution teacher model. 3) Convolutional Shallow Model: Same structure as the convolution student model, but trained on the original hard 0-1 labels. 4) Feed-forward Student Model: A shallow feed-forward network of structure: 1200 linear layer - 1 * 30k feed-forward layer, with soft labels provided by the convolution teacher model. 5) Feed-forward Shallow Model: Same structure as the feed-forward student model, but trained on the original hard 0-1 labels.

The linear bottleneck layer in the models above are utilized to reduce the computational

cost, which is also applied in [3].

Figure 6.3 shows the spectrum of the EDJMs for the models described above. The very complex and deep teacher model achieves significantly better performance than the other models as well as highest normalized singular values for the major components. The student models, which benefit from the soft labels provided by the teacher models, get better performance and higher normalized singular values than the shallow models with the same architecture trained on the 0-1 hard targets. Once more, we observe gaps in major component singular values that correlate with the performance gaps between convolution networks and feed-forward networks. It appears that the deep teacher model is able to learn the most complex model, and that the student models trained to mimic the teacher model are able to learn more complex functions than those learned by shallow models of the same architecture that were trained on the original 0-1 hard targets.

6.3.4 EDJM for Networks with Dropout

Dropout [119] is a widely-used technique that disentangles the dependencies among neural network units by randomly deactivating each neural unit with certain probability. As neural units are dropped out randomly for each mini-batch, dropout effectively provides an efficient way of approximately combining a large collections of neural networks with different structures, and thus often improves the performance of the neural networks.

Figure 6.4 shows that on both MNIST and CIFAR-10, and two different network structures, the dropout networks consistently have higher values on the major components of EDJMs as well as higher performance.

The increase of the major components singular values for dropout networks may result from the independence over hidden units created by dropout. As neural units are dropped-out at random, the correlation among weights is reduced, forcing rows of weight matrices to be more independent of each other. We observe from the EDJMs, which are products of individual weight matrices, that the normalized singular values for major components are higher.

6.3.5 EDJM for an Ensemble of Networks

Figure 6.5 shows the normalized singular values of the EDJM for three DNNs trained on CIFAR-10, and for an ensemble of those DNNs. As shown before, deeper DNNs are more accurate: the 4-DNN has 56.34% accuracy, the 3-DNN has 56.05%, and the 2-DNN has only 54.23%. Also, the normalized singular value scores correlate with the accuracy of the individual models: the 4-DNN has the highest score and the 2-DNN the lowest score.

But what about the ensemble? As expected it has higher accuracy than the individual models (57.60%), but its normalized score is less than the score of even the 2-DNN. This is exactly what we should see if the normalized singular value score is a measure of the relative complexity of the learned functions. Averaging ensembles increase accuracy by reducing variance — an ensemble is *lower* complexity than the individual models contained in it. Averaging suppresses the parts of the learned functions where the models disagree, while emphasizing the regions where the models agree. The net result is that the strongest singular values in the ensemble increase relative to the individual models, while the weaker singular values decrease, both of which lower the score. This result for different model types such as 2-DNNs, 3-DNNs, 4-DNNs and an ensemble of these DNNs provides evidence that the EDJM can be used as an architecture-independent measure of the relative complexity of learned functions.

6.4 Boosting Performance with EDJMs

All the results shown above always find that the better performing models tend to have higher normalized singular values on the major components of EDJMs. To further investigate this and take advantage of the predictive properties of the EDJM for the success in neural network training, we introduce a regularizer on the singular values of EDJMs. We wish to encourage the major singular values of the EDJM to be high relative to the largest singular value, and this can be done by utilizing the following term:

$$r(X, \lambda) = -\lambda \sum_{j=0}^{d_{out}-1} \sum_{\sigma \in S_\theta(X,j)} \log \sigma, \tag{6.13}$$

where λ is a hyper-parameter for the regularization term.

Recall that $S_\theta(X, j)$ denotes all the singular for EDJM conditioned on data input X and jth output. The proposed regularizer is the sum of the log of the singular values of EDJM, which enforces the singular values to be all high due to the diminishing return property of the log function.

While $r(X, \lambda)$ is differentiable, the computational cost is extremely expensive for training neural networks. This is because: (1) X can be as large as the training set to get an accurate estimate of the singular values; (2) even for approximation, where X is as small as a mini-batch, the extra cost to run singular value decomposition (required to calculate the gradients) of EDJMs every mini-batch is appreciable; and (3) unlike standard back-propagation, the gradient with respect to a certain weight matrix depends on all weight matrices before and after the layer of interest.

To utilize our observations about the spectrum of EDJMs in a more tractable manner, we instead place a regularization term on the singular values of each of the layer weights: $r_{layer}(W_l, \lambda) = -\lambda \sum_{\sigma \in S_{layer}(W_l)} \log \sigma$, where $S_{layer}(W_l)$ denotes the singular values of W_l.

As observed in Section 6.3.4, dropout effectively boosts the performance as well as increasing the normalized singular values of the major components of EDJMs by modifying the output of every layer. Accordingly, we expect that putting $r_{layer}(W_l, \lambda)$ on the layer weights will affect singular values of EDJMs. Suppose we consider the ReLU function as a stochastic selector on the rows of the weight matrix, by keeping certain rows while "wiping out" the others by setting them to be zero. If the rows of the weight matrix are already quite dependent, or there is only one dominating singular value for the weight matrix, two subsets of the rows selected by ReLU also tend to be dependent on each other. Conversely, two subsets of independent rows also tend to be independent. For a one hidden layer network, the independence among the modified weight matrices is a good indicator to the independence of rows of EDJMs, which is a sign of high normalized singular values of major components. Such intuition can be carried over to deeper networks as well, as we show in our empirical results. Though such approach is an approximation and not mathematically related to $r(X, \lambda)$, we

will show that $r_{layer}(W_l, \lambda)$ can be easily implemented and works well in practice

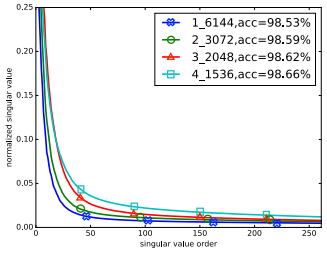
To incorporate $r_{layer}(W_l, \lambda)$ into fully-connected layers, we introduce SVD layers. For input h_{l-1}, instead of a single matrix multiplication $W_l h_{l-1}$, the SVD layer consists of a series of 3 matrices and thus we have $U_l diag(S_l) V_l h_{l-1}$, where $U_l diag(S_l) V_l = SVD(W_l)$, so that U_l and V_l are both orthonormal, S_l contains the singular values of W_l, and $diag(*)$ represents the diagonal matrix with the input vector on the diagonal. In terms of network structure, replacing the ordinary fully-connected layer with dimensions $|h_l| \times |h_{l-1}|$, the SVD layer consists of 3 sub-layers: 1) U layer: a fully connected layer with dimensions $|h_l| \times \min(|h_l|, |h_{l-1}|)$; 2) S layer: a layer with a weight vector of length $\min(|h_l|, |h_{l-1}|)$; 3) V layer: a fully connected layer with dimensions $\min(|h_l|, |h_{l-1}|) \times |h_{l-1}|$. Since we decompose one fully-connected layer into three linear layers, the computational cost for training is about 3 times more than ordinary. However, for inference, we can merge the three linear matrices back together, and there is no extra cost. As the S layer corresponds to the singular values of W_l, we can regularize S layer directly and easily: $r_{layer}(W_l = U_l diag(S_l) V_l, \lambda) = -\lambda \sum_{\sigma \in S_l} \log \sigma$.

To enforce U_l and V_l to be ortho-normal during training, we first update U_l, S_l and V_l to be U_l', S_l', and V_l' according to the gradients, and then set $U_l diag(S_l) V_l = SVD(U_l' diag(S_l') V_l')$. To save the extra computation introduced by the singular value decomposition operation, such projection can be done less frequently. In practice, we do this projection every epoch, which seems to work well.

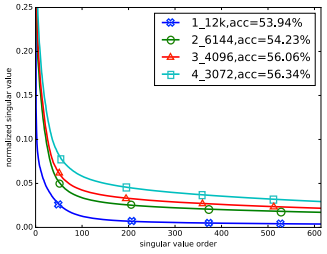
Results shown in Table 6.1 shows improved performance on test set with SVD layers on MNIST, CIFAR-10, and TIMIT datasets. For MNIST and CIFAR-10 in particular, we analyze the spectrum of the EDJMs (see Figure 6.6), and find higher normalized singular values for the major components, which supports our intuition about putting the log-loss spectrum regularization on layer weights. On TIMIT, the spectrally regularized model also had higher accuracy than that trained with dropout. Interestingly, even though spectral regularization increased the EDJM score more than dropout, the resulting models had lower accuracy than dropout on MNIST and CIFAR-10.

Dataset	Structure	Feed-for.	Dropout	Spec. Reg.
MNIST	2 * 1k	98.48%	98.70%	98.66%
CIFAR-10	3 * 4k	56.93%	57.50%	57.43%
TIMIT	3 * 2k	77.77%	78.10%	78.78%

Table 6.1: Test set accuracy (phone accuracy for TIMIT) on MNIST, CIFAR-10 and TIMIT datasets, comparing the baseline feed-forward network with either adding dropout or the spectrum regularization on singular values to the same network structure.



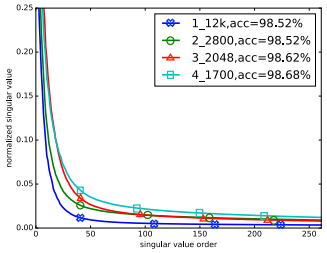
(A) MNIST, Fixed # Hidden Units



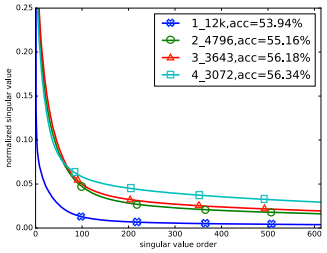
(B) CIFAR-10, Fixed # hidden units

# layers	MNIST score / acc	CIFAR-10 score / acc
1	4.25 / 98.53%	5.99 / 53.94%
2	5.49 / 98.59%	13.94 / 54.23%
3	7.35 / 98.62%	16.58 / 56.06%
4	7.70 / 98.66%	21.40 / 56.34%

(C) Scores for Fixed # Hidden Units (Eq 6.12)



(D) MNIST, Fixed # Parameters

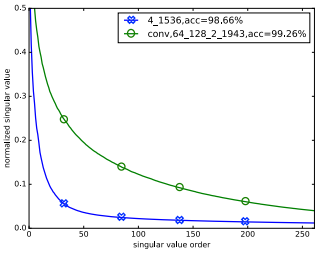


(E) CIFAR-10, Fixed # Parameters

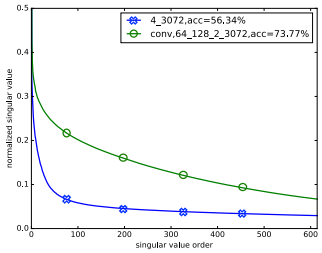
# layers	MNIST score / acc	CIFAR-10 score / acc
1	4.04 / 98.52%	5.99 / 53.94%
2	5.55 / 98.52%	18.64 / 55.16%
3	7.55 / 98.62%	20.39 / 56.18%
4	7.65 / 98.68%	21.40 / 56.34%

(F) Scores for Fixed # Parameters

Figure 6.1: Spectra of EDJMs of feed-forward networks with various number of layers. Validation set accuracy is reported in the legends of each figure and in the tables (D) and (E) (same for following figures and tables). (A) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of total hidden units equal to 6k on MNIST. Each spectrum consists of singular values normalized by the largest singular value (i.e., the curve starts at 1.0 on y-axis), sorted in decreasing order, and then averaged over 10 output classes. We set the maximum value of y-axis to be 0.25 for the purpose of better visual display. (B) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of total hidden units equal to 12k on CIFAR-10. (C) Scores for the spectra plotted in (A) and (B). (D) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of parameters equal to 10 million on MNIST. (E) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of parameters equal to 46 million on CIFAR-10. (F) Scores of the spectra plotted in (D) and (E).



(A) MNIST, Conv. v.s. Feed-forward

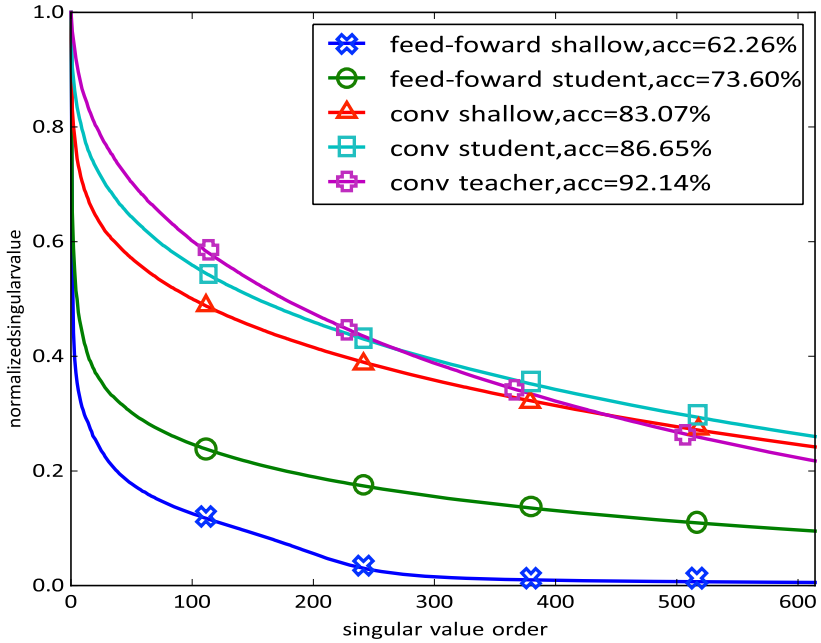


(B) CIFAR-10, Conv. v.s. Feed-forward

Model	MNIST score / acc	CIFAR-10 score / acc
Feed-for.	7.70 / 98.66%	21.40 / 56.34%
Conv.	21.55 / 99.26%	59.65 / 73.77%

(C) Scores for Conv. v.s. Feed-forward

Figure 6.2: Spectra of EDJMs of convolution networks compared to feed-forward networks. (A) Spectra of EDJMs of a 4 * 1536 feed-forward network compared with a 64 filter 3 * 3 - 2 * 2 max pooling - 128 filter 3 * 3 - 2 * 2 max pooling - 2 * 1943 network on MNIST dataset. (B) Spectra of EDJMs of a 4 * 3072 feed-forward network compared with a 64 filter 3 * 3 - 2 * 2 max pooling - 128 filter 3 * 3 - 2 * 2 max pooling - 2 * 3072 feed-forward network. (C) Scores for the spectra plotted in (A) and (B).

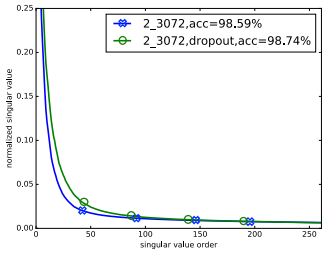


(A) CIFAR-10, Model Compression

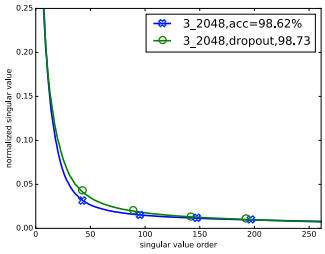
Model	Score	Acc
Feed-forward shallow	33.7	62.3%
Feed-forward student	73.6	73.6%
Convolutional shallow	147.8	83.1%
Convolutional student	162.7	86.7%
Convolutional teacher	172.7	92.1%

(B) Scores for Model Compression Results

Figure 6.3: Effects of model compression training on spectra of EDJMs. (A) shows the spectra of EDJMs for different models in model compression training on CIFAR-10. (B) Scores for (A).



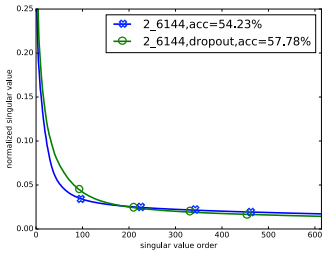
(A) MNIST Dropout 2 * 3k Feed-forward Net



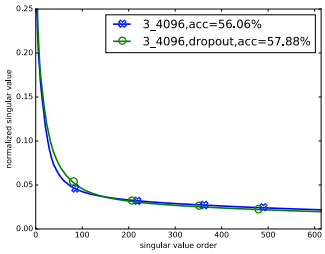
(B) MNIST Dropout 3 * 2k Feed-forward Net

Model	MNIST score / acc
2 layer	5.49 / 98.59%
2 layer dropout	7.15 / 98.74%
3 layer	7.55 / 98.62%
3 layer dropout	8.19 / 98.73%

(C) MNIST Scores for Dropout v.s. Non-dropout



(D) CIFAR-10 Dropout 2 * 6k Feed-forward Net

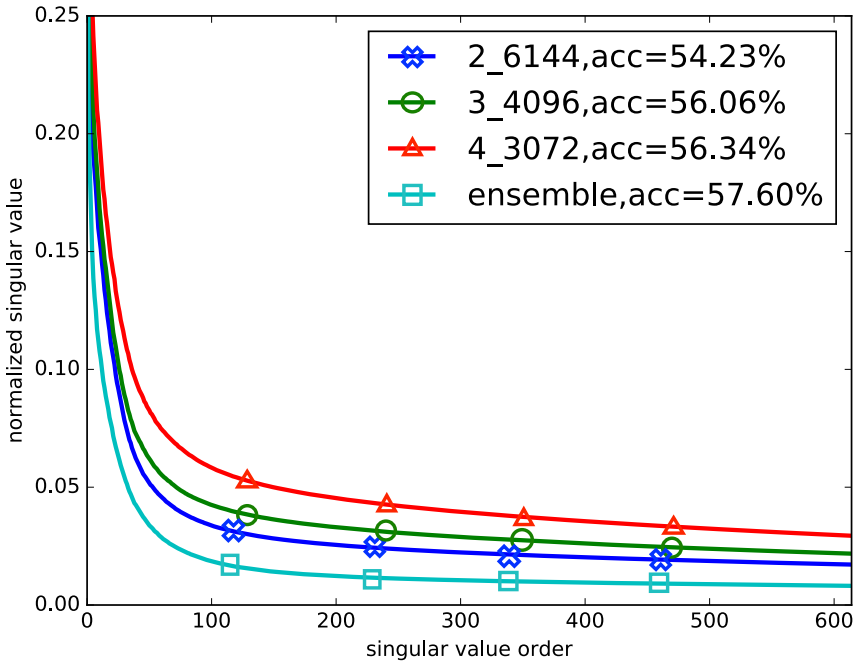


(E) CIFAR-10 Dropout 3 * 4k Feed-forward Net

Model	CIFAR-10 score / acc
2 layer	13.94 / 54.23%
2 layer dropout	16.33 / 57.78%
3 layer	16.58 / 56.06%
3 layer dropout	17.50 / 57.88%

(F) CIFAR-10 Scores for Dropout v.s. Non-dropout

Figure 6.4: Effects of dropout on the spectra of EDJMs. (A) and (B) shows the spectra of EDJMs for models of two network structures feed-forward 2 * 3k and 3 * 2k respectively on MNIST. (C) and (D) shows the spectra of EDJMs for models of two network structures feed-forward 2 * 6k and 3 * 4k respectively on CIFAR-10. (E) Table of scores for the spectra plotted in (A), (B), (C) and (D).

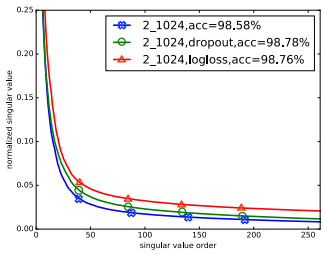


(A) CIFAR-10, Ensemble of Networks

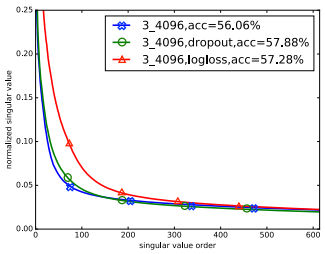
Model	Score	Acc
Feed-forward 2 layer	13.94	54.23%
Feed-forward 3 layer	16.58	56.06%
Feed-forward 4 layer	21.40	56.34%
Ensemble	8.75	57.60%

(B) Scores for Ensemble Results

Figure 6.5: Spectra of EDJMs for ensemble of networks. (A) shows the spectra of 3 different feed-forward networks and their ensemble on CIFAR-10. (B) Scores for spectra plotted in (A).



(A) MNIST Spec. Regularization



(B) CIFAR-10 Spec. Regularization

Model	MNIST score / acc	CIFAR-10 score / acc
Feed-for.	7.00 / 98.58%	16.58 / 56.06%
Dropout	7.51 / 98.78%	17.50 / 57.88%
Spec. Reg.	8.99 / 98.76%	26.44 / 57.28%

(C) Scores for Spec. Regularization

Figure 6.6: The effect of layer weight spectrum regularizer on spectra of EDJMs. (A) Spectra of baseline feed-forward network, dropout network and spectrum regularized network with same network structure (feed-forward 2 * 1k) on MNIST. (B) Spectra of baseline feed-forward network, dropout network and spectrum regularized network with same network structure (feed-forward 3 * 4k) on CIFAR-10. (C) Table of scores for the curved plotted in (A) and (B).

Chapter 7

JUMPOUT: IMPROVED DROPOUT FOR RELU DEEP NETWORKS

While deep neural networks (DNNs) are successful on a wide variety of tasks [102, 99], they are often able to fit training data perfectly, resulting in an overfitting problem and thereby weakening the generalization performance on unseen data. Dropout [118, 49] is a simple yet effective technique to mitigate such problems by randomly setting the activations of hidden neurons to 0, a strategy that reduces co-adaptation amongst neurons. Dropout applies to any layer in a DNN without causing significant additional computational overhead.

Dropout, however, has several drawbacks. Firstly, dropout rates, constituting extra hyper-parameters at each layer, need to be tuned to get optimal performance. Too high a dropout rate can slow the convergence rate of the model, and often hurt final performance. Too low a rate yields few or no improvements on generalization performance. Ideally, dropout rates should be tuned separately for each layer and also during various training stages. In practice, to reduce computation, we often tune a single dropout rate and keep it constant for all dropout layers and throughout the training process.

If we treat dropout as a type of perturbation on each training sample, it acts to generalize the DNN to noisy samples having that specific expected or typical amount of perturbation (due to the fixed dropout rate) with high probability. The fixed rate means noisy samples having less perturbation, i.e., those potentially more likely to be closer to the original samples and thus that are potentially more helpful to improve generalization, are much less likely to occur. Also, when a constant dropout rate is applied to layers and samples having different fractions of activated neurons, the effective dropout rate (i.e., the proportion of the activated neurons that are deactivated by dropout) varies, which might result in immoderate

perturbation for some layers and samples and insufficient perturbation for others.

Another deficiency of dropout lies in its incompatibility with batch normalization (BN) [50] (more empirical evidence of this is shown in Section 7.2.3). As dropout randomly shuts down activated neurons, it needs to rescale the undropped neurons to match the original overall activation gain of the layer. Unfortunately, such rescaling breaks the consistency of the normalization parameters required between training and test phases[1] and may cause poor behavior when used with BN. Since BN, and its variants [4, 129, 142], has become an indispensable component of modern DNN architectures, dropout is itself often dropped out in the choice between these two non-complementary options, and has recently become less popular.

7.0.1 Our Approach

We propose three simple modifications to dropout in order to overcome the aforementioned drawbacks, leading to an improved method we call "jumpout." Our approach is motivated by three observations about how dropout results in improved generalization performance for DNNs with rectified linear unit (ReLU) activations, which covers a frequently used class of DNNs.

Firstly, we note that any DNN with ReLU is a piecewise linear function which applies different linear models to data points from different polyhedra defined by the ReLU activation patterns [86, 134, 132]. Based on this observation, applying dropout to a training sample randomly changes its ReLU activation patterns and hence the underlying polyhedral structure and corresponding linear models. This means that each linear model is trained not only to produce correct predictions for data points in its associated polyhedron, but also is trained to work for data points in nearby polyhedra; what precisely "nearby" means depends on the dropout rate used. This partially explains why dropout improves generalization performance. The problem, however, is that with a fixed dropout rate, say p, and on a layer with n units, the typical number of units dropped out is np as that is the mode of a

[1]Dropout usually happens only during the training but not the test phase, since using it for testing requires averaging the results of multiple dropout inferences on each training sample, which is costly and may introduce greater prediction variance.

Binomial distribution with parameter p. It is relatively rare that either very few (closer to zero) or very many (closer to n) units are dropped out. Thus, with high probability, each linear model is smoothed to work on data points in polyhedra at a typical distance np away. The probability of smoothing over closer distances is much smaller, thus failing to achieve the goal of local smoothness.

In jumpout, by contrast, p rather than being fixed is itself a random variable; we sample p from a distribution that is monotone decreasing (e.g., a truncated half-Gaussian). This achieves the property that Pr(i units dropping out) $\geq$ Pr(i+1 units dropping out) for all $i \in \{1, 2, \ldots, n\}$. That is, a smaller dropout rate has a higher probability of being chosen. Hence, the probability of smoothing polyhedra to other points decreases as the points move farther away.

Secondly, we notice that in dropout, the fraction of activated neurons in different layers, for different samples and different training stages, can be different. Although we are using the same dropout rate, since dropping out neurons that are already quiescent by ReLU changes nothing, the effective dropout rate, i.e., the fraction of the activated neurons that are dropped, can vary significantly. In jumpout, we adaptively normalize the dropout rate for each layer and each training sample/batch, so the effective neural-deactivation rate applied to the activated neurons are consistent over different layers and different samples as training proceeds.

Lastly, we address the incompatibility problem between dropout and BN by rescaling the outputs of jumpout in order to keep the variance unchanged after the process of neural deactivation. Therefore, the BN layers learned in the training phase can be directly applied in the test phase without an inconsistency, and we can reap the benefits of both dropout and BN when training a DNN.

In our implementation, similar to dropout, jumpout also randomly generates a 0/1 mask over the hidden neurons to drop activations. It does not require any extra training, can be easily implemented and incorporated into existing architectures with only a minor modification to dropout code. In our experiments on a broad range of benchmark datasets including

CIFAR10, CIFAR100, Fashion-MNIST, SVHN, STL10 and ImageNet-1k, jumpout shows almost the same memory and computation costs as the original dropout, but significantly and consistently outperforms dropout.

7.0.2 Related Work

Jumpout is not the first approach to address the fixed dropout rate problem. Indeed, recent work has proposed different methods to generate adaptive dropout rates. [5] proposed "standout" to adaptively change the dropout rates for various layers and training stages. They utilized a binary belief network and trained it together with the original network to control the dropout rates. [156] further extend the model so that adaptive dropout rates can be learned for different neurons or group of neurons. [149] showed that the Rademacher complexity of a DNN is bounded by a function related to the dropout rate vectors, and they proposed to adaptively change dropout rates according to the Rademacher complexity of the network. In contrast to the above methods, jumpout does not rely on additional trained models: it adjusts the dropout rate solely based on the ReLU activation patterns. Moreover, jumpout introduces negligible computation and memory overhead relative to the original dropout methods, and can be easily incorporated into existing model architectures.

[135] showed that dropout has a Gaussian approximation called Gaussian dropout and proposed to optimize the Gaussian dropout directly to achieve faster convergence. The Gaussian dropout was also extended and studied from the perspective of variational methods. [63] generalized Gaussian dropout and proposed variational dropout, where they connected the global uncertainty with the dropout rates so that dropout rates can be adaptive for every neuron. [84] further extended variational dropout to reduce the variance of the gradient estimator and achieved sparse dropout rates. Our method is significantly different from the line of work in variational dropout: 1) we do not assume any continuous noise applied to inputs; 2) our goal is not the posterior distribution of model weights; 3) we do not optimize any variational objective or require any Bayesian inference.

Other recent variants of dropout include Swapout [115], which combines dropout with

random skipping connections to generalize to different neural network architectures, and Fraternal Dropout [157], which trains two identical DNNs using different dropout masks to produce the same outputs and tries to shrink the gap between the training and test phases of dropout. Jumpout involves orthogonal and synergistic contributions to most of the above methods, and targets other dropout problems. Indeed, jumpout can be applied along with most other previous variants of dropout.

7.1 ReLU Deep Neural Networks are comprised of local linear models

We study a feed-forward deep neural networks of the form:

$$\hat{y}(x) = W_m \psi_{m-1}(W_{m-1}\psi_{m-2}(\dots\psi_1(W_1 x))), \tag{7.1}$$

where W_j is the weight matrix for layer j, ψ_j is the corresponding activation function (ReLU in this work), $x \in X$ is an input data point of d_{in} dimensions and $\hat{y}(x)$ is the network's output prediction of d_{out} dimensions, e.g., the logits before applying softmax. We denote the hidden nodes at layer j to be h_j, i.e., $h_j = W_j\psi_{j-1}(W_{j-1}\psi_{j-2}(\dots\psi_1(W_1 x)))$, whose dimensionality is d_j. We represent the nodes after applying the activation function as $\bar{h}_j = \psi(h_j)$.

The above DNN formalization can generalize many DNN architectures used in practice. Clearly, Eqn. (8.1) can represent a fully-connected network of m layers. Note Eqn. (8.1) covers the DNNs with bias terms at each layer since the bias terms can be written in the matrix multiplication as well by introducing dummy dimensions on the input data (append m 1's to input data). Moreover, the convolution operator is essentially a matrix multiplication, where every row of the matrix corresponds to applying a convolutional filter on a certain part of the input, and therefore the resulting weight matrix is very sparse and has tied parameters, and typically has an enormous (compared to input size) number of rows. The average-pooling is a linear operator and therefore representable as a matrix multiplication, and max-pooling can be treated as an activation function. Finally, we can represent the residual network block by appending an identity matrix at the bottom of a weight matrix so that we can retain the input values, and add the retained input values later through another

matrix operation. Therefore, we can also write a DNN with short-cut connections in the form of Eqn. (8.1).

For piecewise linear activation functions such as ReLU, the DNN in Eqn. (8.1) can be written as a piecewise linear function, i.e., the DNN in a region surrounding a given data point x is a linear model having the following form:

$$\hat{y}(x) = W_m W_{m-1}^x \dots W_1^x x = \frac{\partial \hat{y}(x)}{\partial x} x, \tag{7.2}$$

where W_j^x is the equivalent weight matrix after combining the resultant activation pattern with W_j. For instance, suppose we use ReLU activation $\psi_{ReLU}(z) = \max(0, z)$; at every layer, we have an activation pattern for the input $a_j(x) \in \{0,1\}^{d_j}$, and $a_j(x)[p] = 0$ indicates that ReLU sets the unit p to 0 or otherwise preserves the unit value. Then, $\psi_{ReLU}(W_1 x) = W_1^x x$, where W_1^x is modified from W_1 by setting the rows, whose corresponding activation patterns are 0, to be all zero vectors. We can continue such a process to the deeper layers, and in the end we can eliminate all the ReLU functions and produce a linear model as shown in Eqn. (8.3).

In addition, the gradient $\frac{\partial \hat{y}(x)}{\partial x}$ is the weight vector of the linear model. Note that the linear model in Eqn. 8.3 is specifically associated with the activation patterns $\{a_j(x)\}_{j=1}^m$ on all layers for a data input x, which is equal to a set of linear constraints that defines a convex polyhedron containing x. In a DNN with ReLU activations, for every dimension i in layer 1, if $a_1(x)[i] = 1$, we have a linear equation $W_1[i]x > 0$ and otherwise we have $W_1[i]x \leq 0$. As a result, we have d_1 linear constraints for layer 1. Similarly, we can follow the same procedure on layer j with input changed to $W_{j-1}^x \dots W_1^x x$, so we have d_j linear constraints for layer j. Therefore, a DNN with piecewise linear activation functions is a piecewise linear function defined by a number of local linear models (on a set of input data points) and the corresponding convex polyhedra, each represented by a set of linear constraints ($\sum_{j=1}^m d_j$ constraints in specific). An analysis based on a similar perspective can be found in [98].

Although the above analysis can be easily extended to DNNs with general piecewise linear activation functions, we focus on DNNs with ReLU activations in the rest of the chapter for

clarity. In addition to the piecewise linear property, ReLU units are cheap to compute, as is their (sub)gradient, and are widely applicable to many different tasks while achieving good performance [42, 146]. In the following, we will study how dropout improves the generalization performance of a complicated DNN by considering how it generalizes each local linear model to its nearby convex polyhedra. This is easier to analyze and acts as the inspiration for our modifications to the original dropout. We will further elaborate the understandings of dropout based on the above insights of local linear models in the next section.

7.2 Three Modifications to Dropout

7.2.1 Modification I: Monotone Dropout Rate for Local Smoothness

There have been multiple explanations for how dropout improves the performance of DNNs. Firstly, dropout prevents the co-adaptation of the neurons in the network, or in other words, encourages independence and diversity amongst the neurons. Secondly, by randomly dropping a portion of neurons during training, we effectively train a large number of smaller networks, and during test/inference, the network prediction can be treated as an ensemble of the outputs from those smaller networks, and thus enjoys the advantages of using an ensemble such as variance reduction.

Here we provide another perspective for understanding how dropout improves generalization performance by inspecting how it smooths each local linear model described in the previous section. As mentioned above, for a DNN with ReLUs, the input space is divided into convex polyhedra, and for any data point in any convex polyhedron of the final layer (a polyhedron that is not divided further into smaller regions), the DNN behaves exactly as a linear model. For large DNNs with thousands of neurons per layer, the number of convex polyhedra can be exponential in the number of neurons. Hence, there is a high chance that the training samples will be haphazardly situated amongst the different polyhedra, and every training data point is likely to be given its own distinct local linear model. Moreover, it is possible that two nearby polyhedra may correspond to arbitrarily different linear

models, since they are the results of consecutively multiplying a series of weight matrices $W_m W_{m-1}^x \dots W_1^x$ of different x (as shown in Eqn. (8.3)), where each weight matrix W_j^x is W_j with some rows setting to be all-zero according to the activation pattern $a_j(x)$ of a specific data point x. If the activation patterns of two polyhedra differ on some critical rows of the weight matrices, the resulting linear models may in fact differ greatly. Therefore, it is possible for the linear model of one polyhedron to work only for one or a few training points strictly within the polyhedron, and it may fail when applied to nearby test data points (i.e., a lack of smoothness). This can make the DNN fragile and perform unstably on new data, thus weakening its generalization performance.

Given the problems of dropout mentioned in Section 7.0.1, we propose to sample a dropout rate from a truncated half-normal distribution (to get a positive value), which is the positive part of an ordinary Gaussian distribution with mean zero. In particular, we firstly sample $p \sim \mathcal{N}(0, \sigma)$ from a Gaussian distribution, and then take the absolute value $|p|$ as the dropout rate. We further truncate $|p|$ so that $|p| \in [p_{\min}, p_{\max}]$, where $0 \leq p_{\min} < p_{\max} \leq 1$. These determine the lower and upper limits of the dropout rate and are used to ensure the sampled probability is neither too small, which makes jumpout ineffective, nor too large, which may yield poor performance. Overall, this achieves monotone decreasing probability of a given dropout rate as mentioned above. Other distributions (such as a Beta distribution) could also be used for this purpose, but we leave that to future work.

We utilize the standard deviation σ as the hyper-parameter to control the amount of generalization enforcement. By using the above method, smaller dropout rates are sampled with higher probabilities so that a training sample will be more likely to contribute to the linear models of closer polyhedra. In Figure 7.1, we compare a constant dropout rate and a monotone dropout rate. A Gaussian-based dropout rate distribution encourages greater smoothness between local linear models for close polyhedra, but this encouragement diminishes between polyhedra farther away from each other.

7.2.2 Modification II: Dropout Rate adapted to the number of Activated Neurons

The dropout rate for each layer is a hyper-parameter, and as stated above, it controls a form of smoothness amongst nearby local linear models. Ideally, the dropout rates of different layers should be tuned separately to improve network performance. In practice, it is computationally expensive or infeasible to tune so many hyper-parameters. One widely adopted approach is therefore to set the same drop rate for all layers and to tune one global dropout rate.

Using a single global dropout rate is suboptimal because the proportion of active neurons (i.e., neurons with positive values) of each layer at each training stage and for each sample can be dramatically different (see Fig. 7.2). When applying the same dropout rate to different layers, different fractions of active neurons get deactivated, so the effective dropout rate applied to the active neurons varies significantly. Suppose the fraction of active neurons in layer j is $q_j^+ = (\sum_{i=1:d} \mathbb{1}_{h_j[i]>0})/|h_j|$. Since dropping the inactive neurons has no effect (neurons with values ≤ 0 have already been set to 0 by ReLU), the effective dropout rate of every layer is $p_j q_j^+$, where p_j is the dropout rate of layer j. Thus, to better control the behavior of dropout for different layers and across various training stages, we normalize the dropout rate by q_j^+ and use an actual dropout rate of $p'_j = p_j/q_j^+$. By doing so, the hamming distance between the changed activation pattern and the original pattern is more consistent, and we can more precisely achieve the desirable level of smoothing encouragement by tuning the dropout rate as a single hyper-parameter.

7.2.3 Modification III: Rescale Outputs to work with Batch Normalization

In standard dropout, if the dropout rate is p, we scale the neurons by $1/p$ during training and keeps the neuron values unchanged during the test/inference phase. The scaling factor $1/p$ keeps the mean of the neurons the same between training and test; this constitutes a primary reason for the incompatibility between dropout and batch normalization (BN) [50]. Specifically, though the mean of neurons is consistent, the variance can be dramatically different between the training and test phases, in which case the DNN might have unpredictable be-

havior as the BN layers cannot adapt to the change of variance from training to test condition.

We consider one possible setting of combining dropout layers with BN layers where one linear computational layer (e.g., a fully-connected or a convolutional layer without activation function) is followed by a BN layer, then a ReLU activation layer, and then followed by a dropout layer. For layer j, without loss of generality, we may treat the value of a neuron i after ReLU, i.e., $\bar{h}_j[i]$ as a random variable with q_j^+ probability of being 1 and $1 - q_j^+$ probability of being 0. If dropout is not applied, $\bar{h}_j[i]$ then gets multiplied by certain entry in the weight matrix $W_{j+1}[i', i]$, and contributes to the value of the i' neuron of layer $j + 1$. Since we consider any index i and i', we rename the following terms for simplicity: $x_j := \bar{h}_j$, $w_j := W_{j+1}[i', :]$, $y_j := h_{j+1}[i']$. As neuron i' of layer $j + 1$ (before ReLU) then gets fed into a BN layer, we will focus on the change of mean and variance as we add the dropout layer.

Suppose we apply a dropout rate of p_j, then

$$\mathbb{E}[y_j] = [w_j](1 - p_j)q_j^+ \tag{7.3}$$

$$\begin{aligned} Var[y_j] &= \mathbb{E}[y_j^2] - \mathbb{E}[y_j]^2 \\ &= (1 - p_j)\mathbb{E}[(w_j x_j)^2] - (\mathbb{E}[w_j](1 - p_j)q_j^+)^2, \end{aligned} \tag{7.4}$$

where the expectation is taken over all neurons in the same layer. Hence, dropout changes both the scales of the mean and variance of neurons during training. Since the following BN's parameters are trained based on the scaled mean and variance, which however are not scaled by dropout during test/inference (because dropout is not used during testing), the trained BN is not consistent with the test phase. An easy fix of the inconsistency is to rescale the output y_j to counteract dropout's on the scales of mean and variance. In order to recover the original scale of the mean, we should rescale the dropped neurons by $(1 - p_j)^{-1}$. However, the rescaling factor should be $(1 - p_j)^{-0.5}$ instead for recovering the scale of the variance if $\mathbb{E}(y_j)$ is small and thus the second term of the variance can be ignored.

Ideally, we can also take into account the value of $\mathbb{E}[w_j]$, and scale the un-dropped neurons

by

$$\sqrt{\frac{\mathbb{E}[(w_j x_j)^2] - (\mathbb{E}[w_j] q_j^+)^2}{(1-p_j)\mathbb{E}[(w_j x_j)^2] - (\mathbb{E}[w_j](1-p_j) q_j^+)^2}}. \tag{7.5}$$

However, computing information about W_j, which is the weight matrix of layer-j, requires additional computational and memory costs. In addition, such a scaling factor is only correct for the variance of y_j. To make the mean consistent, we should instead use $(1-p_j)^{-1}$ (the original dropout scaling factor). No simple scaling method can resolve the shift in both mean and variance, as the mean rescaling $(1-p_j)^{-1}$ does not solve the variance shift.

When the mean $\mathbb{E}(y_j)$ is large in magnitude, so that the second term in the variance is comparable with the first term, in which case the variance is small, we should use the rescaling factor close to $(1-p_j)^{-1}$, which makes the mean exactly unchanged for training and test. In contrast, when the mean $\mathbb{E}(y_j)$ is small in magnitude and close to 0, the second term in the variance is ignorable, and we should use $(1-p_j)^{-0.5}$ as the rescaling factor, to make the variance unchanged. In practice, it is not efficient to compute $E(y_j)$ during training, so we propose to use a trade-off point $(1-p_j)^{-0.75}$ between $(1-p_j)^{-1}$ and $(1-p_j)^{-0.5}$. In Figure 7.4, we show that $(1-p_j)^{-0.75}$ makes both the mean and variance sufficiently consistent for the cases of using dropout and not using dropout. In Figure 7.3, we compare the performance of the original dropout and dropout using our rescaling factor $(1-p_j)^{-0.75}$, when they are used with and without BN in a convolutional networks. It shows that using dropout with BN can potentially improve the performance, and larger dropout might result in more improvement. However, using the original dropout with BN leads to a significant decrease in the accuracy once increasing the dropout rate over 0.15. In contrast, the performance of dropout using our rescaling keeps improving with increasing dropout rate (until reaching 0.25), and is the best among the four configurations.

7.2.4 Jumpout Layer

We combine the three modifications specifically designed to overcome the drawbacks of the original dropout in our proposed improved dropout, which we call "jumpout" as shown in Alg. 13. Similar to the original dropout, jumpout essentially generates a 0/1 mask for the

Algorithm 13: Jumpout Layer with ReLU

input : h_j, σ, $p_{\max}$, $p_{\min}$

$q_j^+ := (\sum_{i=1:d_j} \mathbb{1}_{h_j[i]>0})/|h_j|$; `// Compute the fraction of activated neurons`

$p \sim \mathcal{N}(0, \sigma), p_j := \min(p_{\min} + |p|, p_{\max})$; `// Sample a Gaussian dropout rate`

$p'_j := p_j/q_j^+$; `// Normalize the dropout rate according to the fraction of activated neurons`

Randomly generate a 0/1 mask z_j for h_j, with probability p'_j to be 0; `// Sample the dropout mask`

$s_j := (1 - p')^{-0.75}$; `// Compute the rescaling factor`

$h'_j := s_j * \text{diag}(z_j)h_j$; `// Rescale the outputs`

return h'_j

input neurons, and randomly drop a portion of the neurons based on the mask. We sample a dropout rate for each layer on each training batch.

Summarizing the novelty of jumpout, instead of using a fixed dropout rate as in the original dropout, jumpout samples from a monotone decreasing distribution as mentioned above to get a random dropout rate. Also, jumpout normalizes the dropout rate adaptively based on the number of active neurons, which enforces consistent regularization and generalization effects on different layers, across different training stages, and on different samples. Finally, jumpout further scales the outputs by $(1-p)^{-0.75}$, as opposed to $(1-p)^{-1}$ during training, in order to trade-off the mean and variance shifts and synergize well with batchnorm operations.

Jumpout requires one main hyper-parameter σ to control the standard deviation of the half-normal distribution, and two auxiliary truncation hyperparameters $(p_{\min}, p_{\max})$. Though $(p_{\min}, p_{\max})$ can also be tuned, they serve to bound the samples from the half-normal distribution; in practice, we set $p_{\min} = 0.01$ and $p_{\max} = 0.6$, which work consistently well over all datasets and models we tried. Hence, jumpout has three hyperparameters, although we only tuned σ and achieved good performance, as can be seen below.

Also, note that here we consider the input h_j to be the features of layer j corresponding to one data point. For a mini-batch of data points, we can either estimate q_j^+ separately for each single data point in the mini-batch or apply the average q_j^+ over data points as the estimate for the mini-batch. In practice, we utilize the latter option as we find that it gives comparable performance to the first while using less computation and memory.

Table 7.1: Ablation study (test accuracy in %) of all the possible combinations of the three modifications (I, II and III) in jumpout, "CIFAR10(s)" refers to the small CNN applied to CIFAR10.

Dataset	CIFAR10(s)	CIFAR10	CIFAR100	STL10
Dropout	86.50	95.23	79.41	81.37
Dropout+I	87.56	96.06	79.89	81.76
Dropout+I+II	90.06	96.35	81.70	82.09
Dropout+II	87.14	95.67	80.24	81.94
Dropout+II+III	89.64	96.74	80.61	82.22
Dropout+III	87.70	96.20	80.59	81.69
Dropout+I+III	87.36	96.45	81.20	82.18
Jumpout	90.24	96.82	82.48	84.02

Jumpout has almost the same memory cost as the original dropout, which is the additional 0/1 drop mask. For computation, jumpout requires counting the number of active neurons, which is insignificant compared to the other layers of a deep model, and sampling from the distribution, which is also insignificant compared to the other computation in DNN training.

7.3 Experiments

In this section, we apply dropout and jumpout to different popular DNN architectures and compare their performance on six benchmark datasets at different scales. In partic-

ular, these DNN architectures include a small CNN with four convolutional layers[2] applied to CIFAR10 [68], WideResNet-28-10 [146] applied to CIFAR10 and CIFAR100 [68], "pre-activation" version of ResNet-20 [44] applied to Fashion-MNIST ("Fashion" in all tables) [143], WideResNet-16-8 applied to SVHN [92] and STL10 [23], and ResNet-18 [42] applied to ImageNet [28, 102].

Table 7.2: Test accuracy (%) of different DNNs trained without dropout/jumpout, with dropout, and with jumpout (10 random trials).

Dataset	CIFAR10(s)	CIFAR10	CIFAR100	Fashion	STL10	SVHN	ImageNet
Original	82.47	94.07	77.98	95.85	75.21	97.39	71.04
Dropout	86.43 ± 0.11	95.21 ± 0.10	79.34 ± 0.07	95.85 ± 0.14	81.09 ± 0.27	98.15 ± 0.05	71.09 ± 0.08
Jumpout	90.18 ± 0.13	96.69 ± 0.08	82.22 ± 0.09	97.13 ± 0.12	83.87 ± 0.24	98.36 ± 0.04	71.43 ± 0.06

For all the experiments about CIFAR and Fashion-MNIST, we follow the standard settings, data preprocessing, augmentation, and hyperparameters used in an existing GitHub repository [3]. On each dataset, we tune the dropout rate and σ in jumpout between $[0.05, 0.60]$ with a step size of 0.05 on a validation set that is 20% of the original training set. On ImageNet, we starts from a pre-trained ResNet18 model[4], and train two copies of it with dropout and jumpout respectively for the same number of epochs. The reason for not starting from random initialized model weights is that training DNNs on ImageNet usually does not have overfitting problem if one follows the standard data augmentation methods used to train most modern models, but both dropout and jumpout are most effective in the case of overfitting. Therefore, we choose to start from the pre-trained model, on which training accuracy is relatively high (but still not overfit and very close to the test accuracy)[5].

[2]The "v3" network from `https://github.com/jseppanen/cifar_lasagne`.

[3]`https://github.com/hysts/pytorch_image_classification`

[4]`https://gluon-cv.mxnet.io/api/model_zoo.html#gluoncv.model_zoo.resnet18_v1b`

[5]In fact, ImageNet is less of an appropriate benchmark to test the performance of dropout, and almost all the modern DNNs for ImageNet do not use dropout, because the training accuracy is always close to the test accuracy during the training process, and there is no overfitting problem needed to be tackled by dropout. We include ImageNet in experiments because of its large size compared to the other datasets

We summarize the experimental results in Table 7.2 which shows that jumpout consistently (i.e., always) outperforms dropout on all datasets and all the DNNs we tested. Moreover, for Fashion-MNIST and CIFAR10 on which the test accuracy is already $> 95\%$, jumpout can still bring appreciable improvements. In addition, on CIFAR100 and ImageNet (on which a great number of DNNs and training methods are heavily tuned), jumpout achieves the improvement that can only be obtained by significantly increasing the model size in the past. These verify the effectiveness of jumpout and its advantage comparing to the original dropout.

In addition, we conduct a thorough ablation study of all the possible combinations of the three proposed modifications, with results reported in Table 7.1. It further verifies the effectiveness of each modification: 1) each modification improves the vanilla dropout; 2) adding any modification to another brings further improvements; and 3) applying the three modifications together achieves the best performance.

We also provide the learning curves and convergence plots of dropout and jumpout equipped DNNs during training in Figure 7.5. In all the figures, "jumpout" applies adaptive dropout rate per mini-batch. Jumpout exhibits substantial advantages over dropout in early learning stages, and reaches reasonably good accuracy much sooner. In the future, it may be possible to find a better learning rate schedule method specifically for jumpout, so it can reach the final performance earlier than dropout.

---

used.

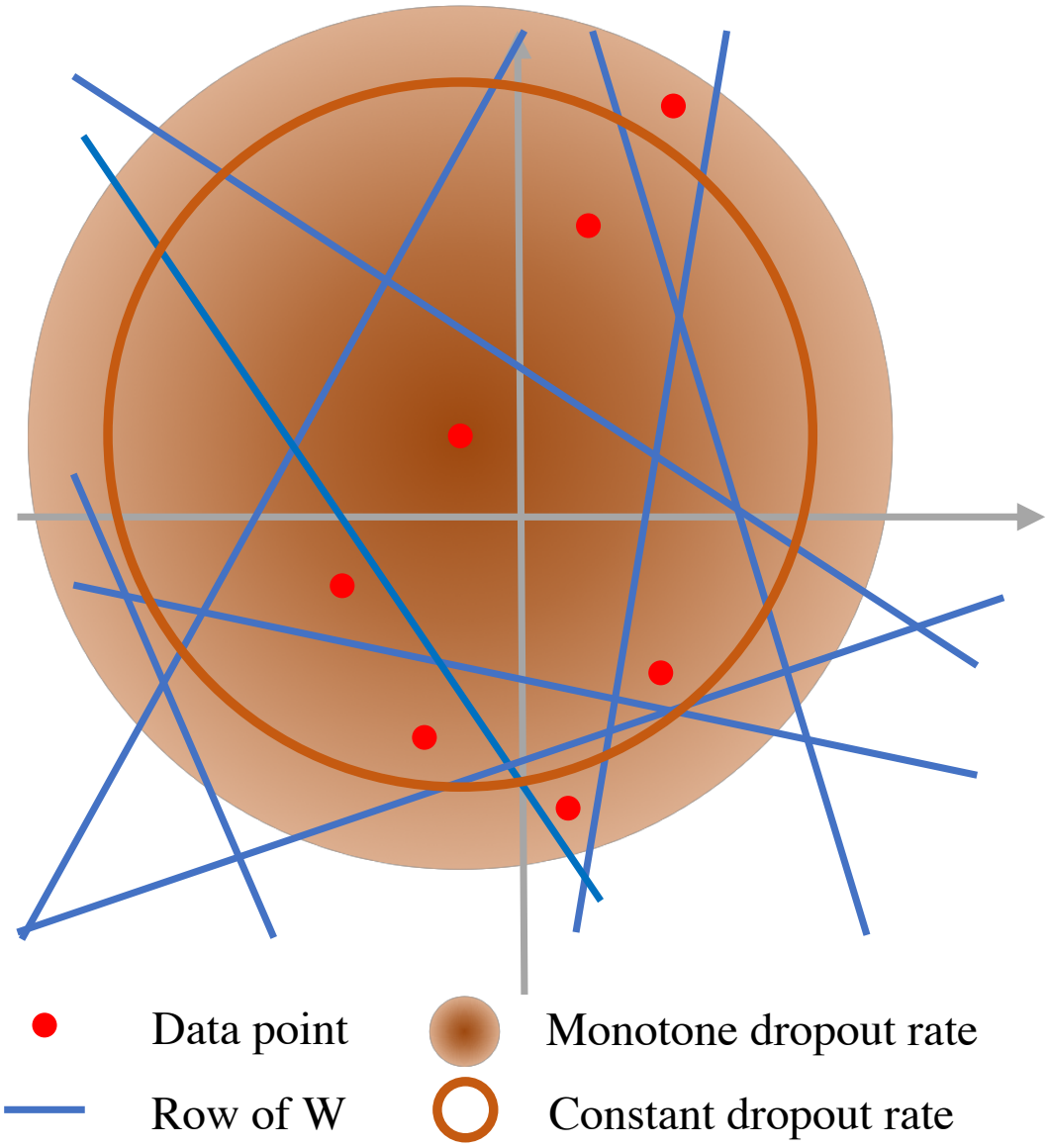


Figure 7.1: DNNs with ReLU partitions the input space into multiple polyhedra (using the blue lines defined by the rows of weight matrix W), and applies a linear model to data points within each polyhedron. Dropout means randomly applying the linear model of a polyhedron to data points in some other polyhedra. When a constant dropout rate is used, the center data point, for example, will be assigned, with high probability, to the linear models of polyhedra at constant distance (the orange $\circ$). By contrast, a monotone dropout rate probability assigns the data point to linear models of closer polyhedra with higher probability than farther polyhedra.

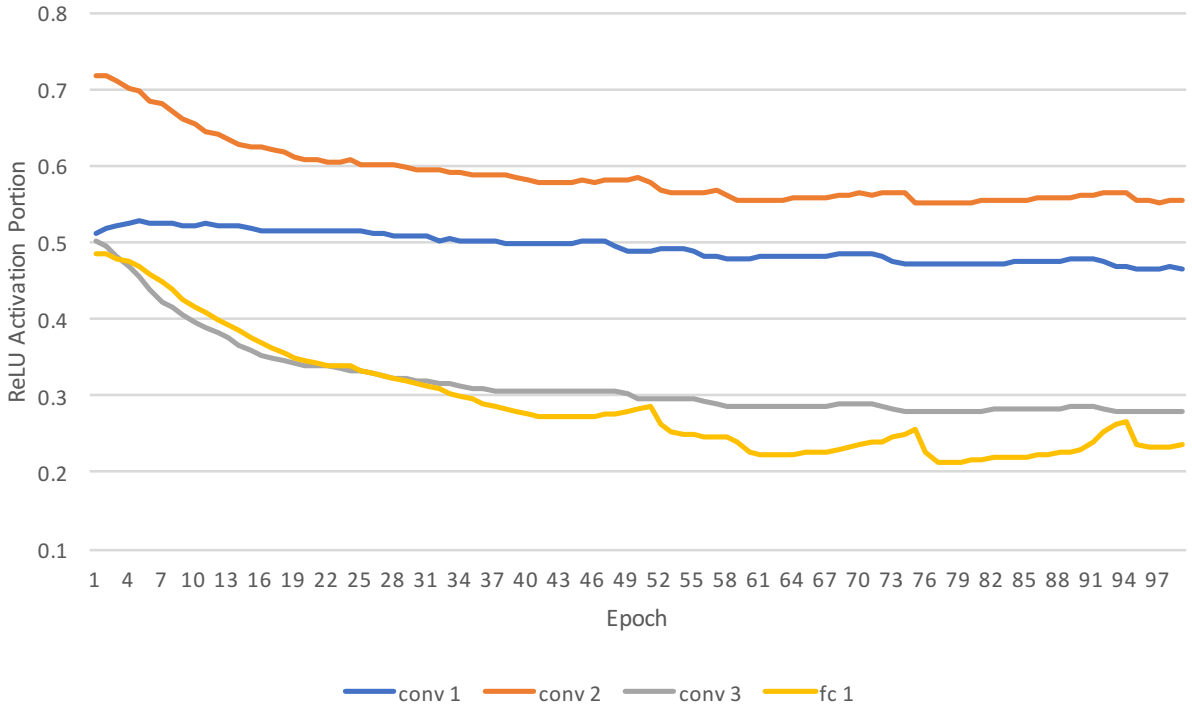


Figure 7.2: Portion of activate neurons on different layers throughout the training process. The network is "CIFAR10(s)" (see Sec. 8.3).

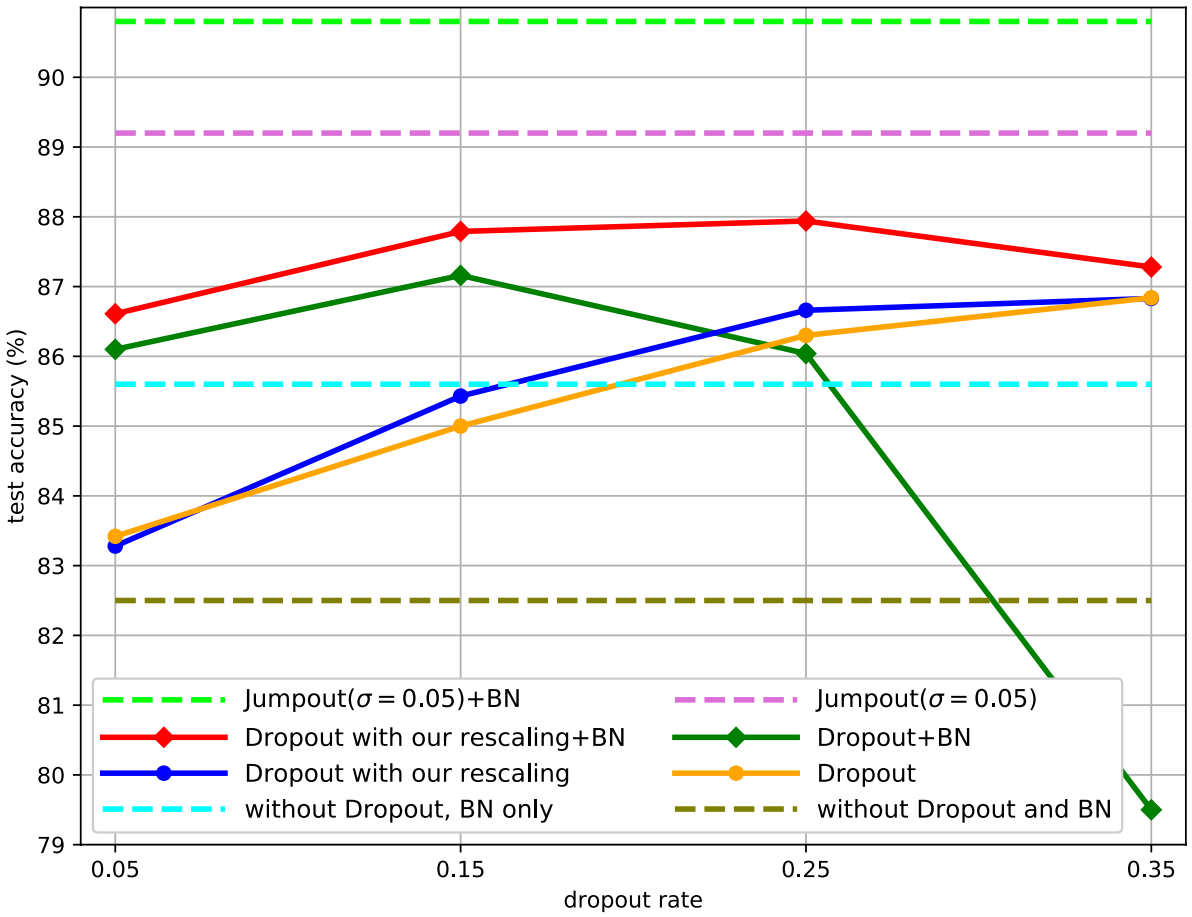
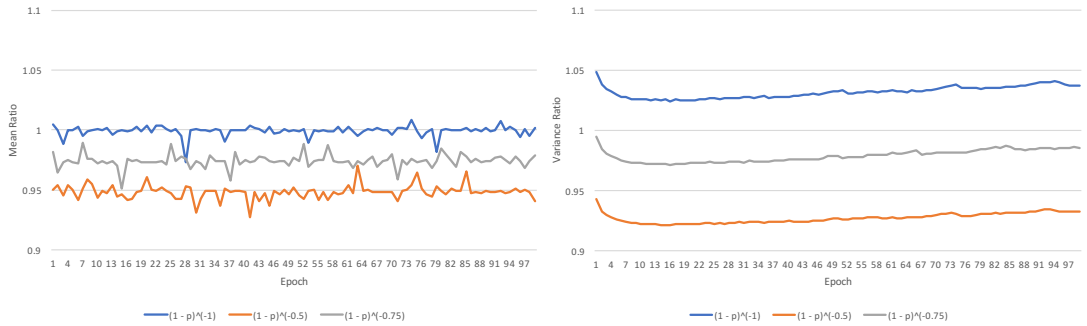
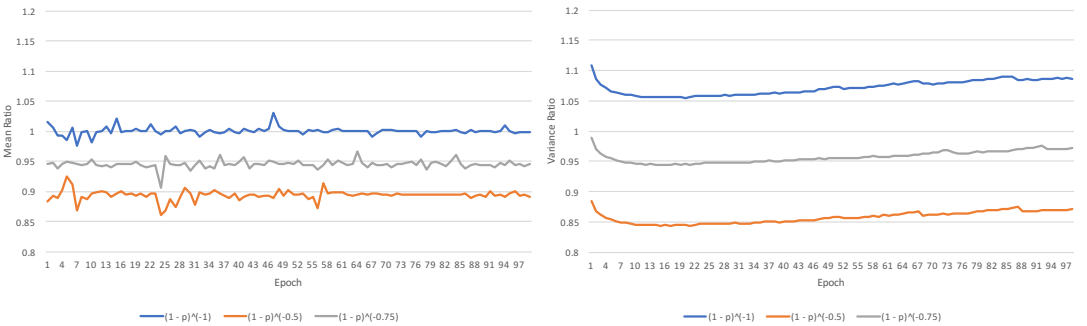


Figure 7.3: Comparison of the original dropout, dropout with our rescaling and jumpout, on their performance (after 150 training epochs) when used with or without batch normalization (BN) in "CIFAR10(s)" network (see Sec. 8.3). Jumpout will be formally introduced in Sec. 7.2.4.



(a) $p = 0.1$



(b) $p = 0.2$

Figure 7.4: Comparison of mean/variance drift when using $(1-p)^{-1}$, $(1-p)^{-0.5}$ and $(1-p)^{-0.75}$ as the dropout rescaling factor applied to y, when $p = 0.1$ and $p = 0.2$. The network is "CIFAR10(s)" (see Sec. 8.3). The left plot shows the empirical mean of y with dropout divided by the case without dropout (averaged over all layers), and the second plot shows the similar ratio for the variance. Ideally, both ratios should be close to 1. As shown in the plots, $(1-p)^{-0.75}$ gives nice trade-offs between the mean and variance rescaling.

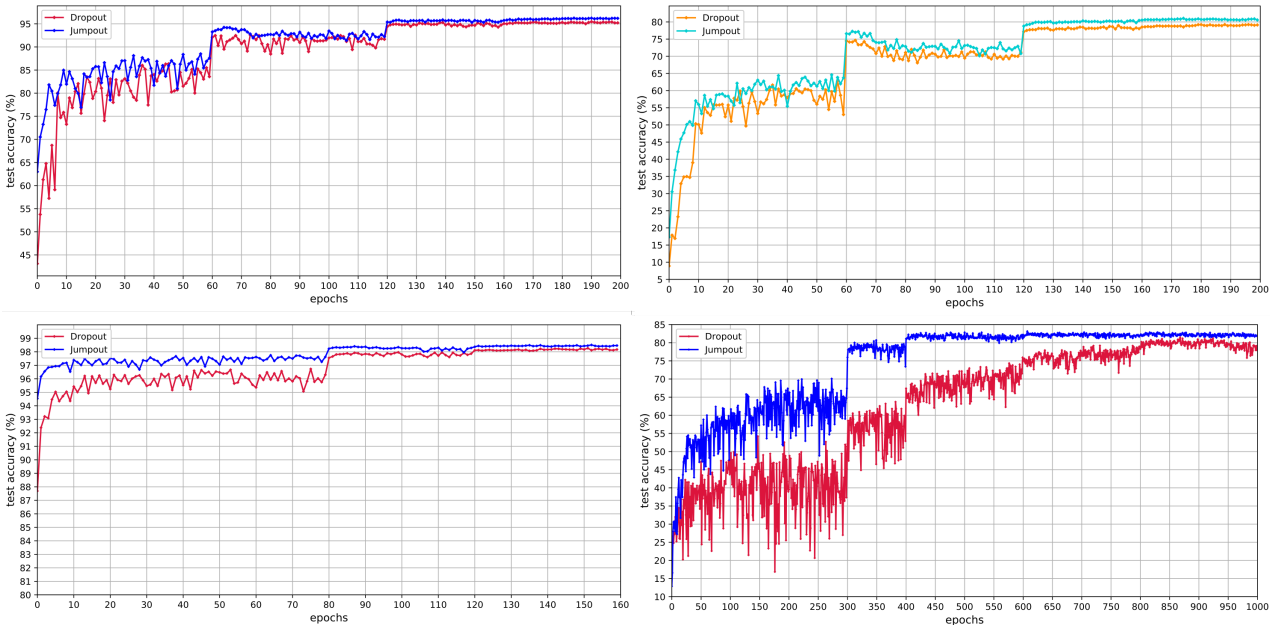


Figure 7.5: **Top Left**: WideResNet-28-10+Dropout and WideResNet-28-10+Jumpout on CIFAR10; **Top Right**: WideResNet-28-10+Dropout and WideResNet-28-10+Jumpout on CIFAR100; **Bottom Left**: WideResNet-16-8+Dropout and WideResNet-16-8+Jumpout on SVHN; **Bottom Right**: WideResNet-16-8+Dropout and WideResNet-16-8+Jumpout on STL10.

Chapter 8

BIAS ATTRIBUTION FOR DEEP NETWORK EXPLANATION

Deep neural networks (DNNs) have produced good results for many challenging problems in computer vision, natural language processing, and speech processing. Deep learning models, however, are usually designed using fairly high-level architectural decisions, leading to a final model that is often seen as a difficult to interpret black box. DNNs are a highly expressive trainable class of non-linear functions, utilizing multi-layer architectures and a rich set of possible hidden non-linearities, making interpretation by a human difficult. This restricts the reliability and usability of DNNs especially in mission-critical applications where a good understanding of the model's behavior is necessary.

The gradient is a useful starting point for understanding and generating explanations for the behavior of a complex DNN. Having the same dimension as the input data, the gradient can reflect the contribution to the DNN output of each input dimension. Not only does the gradient yield attribution information for every data point, but also it helps us understand other aspects of DNNs, such as the highly celebrated adversarial examples and defense methods against such attacks [124].

When a model is linear, the gradient recovers the weight vector. Since a linear model locally approximates any sufficiently smooth non-linear model, the gradient can also be seen as the weight vector of that local linear model for a given DNN at a given data point. For a piecewise linear DNN (e.g., a DNN with activation functions such as ReLU, LeakyReLU, PReLU, and hard tanh) the gradient is exactly the weights of the local linear model[1].

Although the gradient of a DNN has been shown to be helpful in understanding the be-

[1]This is true except when the gradient is evaluated at an input on the boundary of the polyhedral region within which the DNN equals to the local linear model. In such case, subgradients are appropriate.

havior of a DNN, the other part of the locally linear model, i.e., the bias term, to the best of our knowledge, has not been studied explicitly and is often overlooked. If only considering one linear model within a small region, the bias, as a scalar, seems to contain less information than the weight vector. However, this scalar is the result of complicated processing of bias terms over every neuron and every layer based on the activations, the non-linearity functions, as well as the weight matrices of the network. The bias term is data dependent and different samples can have different bias terms in the local linear models. Uncovering the bias's nature could potentially reveal a rich vein of attribution information complementary to the gradient. For classification tasks, it can be the case that the gradient part of the linear model contributes to only a negligible portion of the target label's output probability (or even a negative logit value), and only with a large bias term does the target label's probability becomes larger than that of other labels to result in the correct prediction (see Sec 8.3). In our empirical experiments (Table 8.1), using only the bias term of the local linear models achieves 30-40% of the performance of the complete DNN, thus indicating that the bias term indeed plays a substantial role in the mechanisms of a DNN.

In this chapter, we unveil the information embedded in the bias term by developing a general bias attribution framework that distributes the bias scalar to every dimension of the input data. We propose a backpropagation-type algorithm called "bias backpropagation (BBp)" to send and compute the bias attribution from the output and higher-layer nodes to lower-layer nodes and eventually to the input features, in a layer-by-layer manner. Specifically, BBp utilizes a recursive rule to assign the bias attribution on each node of layer ℓ to all the nodes on layer $\ell-1$, while the bias attribution on each node of layer $\ell-1$ is composed of the attribution sent from the layer below and the bias term incurred in layer $\ell-1$. The sum of the attributions over all input dimensions produced by BBp exactly recovers the bias term in the local linear model representation of the DNN at the given input point. In experiments, we visualize the bias attribution results as images on a DNN trained for image classification. We show that bias attribution can highlight essential features that are complementary with what the gradient-alone attribution methods favor.

8.0.1 Related Work

Attribution methods for deep models are important to complement good empirical performance of DNNs with explanations for how, why, and in what manner do such complicated models make their decisions. Ideally, such methods would render DNNs as glass boxes rather than black boxes. To this end, a number of strategies have been investigated. [114] visualized behaviors of convolutional networks by investigating the gradients of the predicted class output with respect to the input features. Deconvolution [148] and guided backpropagation [117] modify gradients with additional constraints. [85] extended to higher order gradient information by calculating the Taylor expansion, and [11] study the Taylor expansion approach on DNNs with local renormalization layers. [110] proposed DeepLift, which separates the positive from the negative attribution, and features customer designed attribution scores. [120] declare two axioms an attribution method needs to satisfy. It further develops an integrated gradient method that accumulates gradients on a straight-line path from a base input to a real data point and uses the aggregated gradients to measure the importance of input features. Class Activation Mapping (CAM) [152] localizes the attribution based on the activation of convolution filters, and can only be applied to a fully convolutional network. Grad-CAM [106] relaxes the all-convolution constraints of CAM by incorporating the gradient information from the non-convolutional layers. [**?**] explain any classifier by utilizing an additional interpretable model locally around the target data point. They frame the attribution task as a submodular optimization problem and aims to identify the explanations in a non-redundant way. [**?**] propose a unified attribution method that generalizes many attribution methods that are additive over features. They propose a Shapely value based score as a unified measurement. All the work mentioned above utilizes information encoded in the gradients in some form or another, but none of them explicitly investigates the importance of the bias terms, which is the focus of this work. Some of them, e.g. [110] and [120], consider the overall activation of neurons in their attribution methods, so the bias terms are implicitly taken into account, but are not independently studied. Moreover,

some previous work (e.g. CAM) focuses on the attribution for specific network architectures such as convolutional networks, while our approach applies generally to any piece-wise linear DNN, convolutional or otherwise.

8.1 Background and Motivation

We can write the output $f(x)$ of any feed-forward deep neural network in the following form:

$$\begin{aligned} f(x) = W_m \psi_{m-1}(W_{m-1}\psi_{m-2} \\ (\dots \psi_1(W_1 x + b_1)\dots) + b_{m-1}) + b_m, \end{aligned} \tag{8.1}$$

where W_i and b_i are the weight matrix and bias term for layer i, ψ_i is the corresponding activation function, $x \in X$ is an input data point of d_{in} dimensions, $f(x)$ is the network's output prediction of d_{out} dimensions, and each hidden layer i has d_i nodes. We rule out the last softmax layer from the network structure; for example, the output $f(x)$ may refer to logits (which are the inputs to a softmax to compute probabilities) if the DNN is trained for classification tasks.

The above DNN formalization generalizes many widely used architectures. Clearly, Eq. (8.1) can represent a fully-connected network of m layers. Moreover, the convolution operation is essentially a matrix multiplication, where every row of the matrix corresponds to applying a filter from convolution on a certain part of the input, and therefore the resulting weight matrix has tied parameters, is very sparse, and typically has a very large (compared to the input size) number of rows. Average-pooling is essentially a linear operation and therefore is representable as matrix multiplication, and max-pooling can be treated as an activation function. Batchnorm [51] is a linear operation and can be combined into the weight matrix. Finally, we can represent a residual network [41] block by appending an identity matrix at the bottom of a weight matrix so that we can keep the input values, and then add the kept input values later via another matrix operation.

8.1.1 Piecewise Linear Deep Neural Networks

We will focus on DNNs with piecewise linear activation functions, which cover most of the recently successful neural networks in a variety of application domains. Some widely used piecewise linear activation functions include the ReLU, leaky ReLU, PReLU, and the hard tanh functions. A general form of a piecewise linear activation function applied to a real value z is as follows:

$$\psi(z) = \begin{cases} c^{(0)} \cdot z, & \text{if } z \in (\eta_0, \eta_1] \\ c^{(1)} \cdot z, & \text{if } z \in (\eta_1, \eta_2] \\ \cdots, & \cdots \\ c^{(h-1)} \cdot z, & \text{if } z \in (\eta_{h-1}, \eta_h) \end{cases} \tag{8.2}$$

In the above, there are h linear pieces, and these correspond to h predefined intervals on the real axis. We define the activation pattern $\phi(z)$ of z as the index of the interval containing z, which can be any integer from 0 to $h-1$. Both $\psi(z)$ and $\phi(z)$ extend to element-wise operators when applied to vectors or high dimensional tensors.

As long as the activation function is piecewise linear, the DNN is a piecewise linear function and is equivalent to a linear model at and near each input point x [86, 132]. Specifically, each linear model piece of the DNN (associated with an input point x) is:

$$\begin{aligned} f(x) =& \prod_{i=1}^{m} W_i^x x + \left(\sum_{j=2}^{m} \prod_{i=j}^{m} W_i^x b_{j-1}^x + b_m \right) \\ =& \frac{\partial f(x)}{\partial x} x + b^x. \end{aligned} \tag{8.3}$$

This holds true for all the possible input points x on the linear piece of a DNN. We will give a more general result later in Lemma 12. Note W_i^x and b_i^x in the above linear model are modified from W_i and b_i respectively and have to fulfill

$$x_{i+1} = \psi_i(W_i x_i + b_i) = W_i^x x_i + b_i^x, \tag{8.4}$$

where x_i is the activation of layer i (x_1 is the input data) and b_i^x is an x_i-dependent bias vector. In the extreme case, no two input training data points share the same linear model

in Eq. (8.3). In this case, the DNN can still be represented as a piecewise linear model and each local linear model is only applied to one data point.

Given x_i, W_i^x and b_i^x can be derived from W_i and b_i according to the activation pattern vector $\phi(W_i x_i + b_i)$. In particular, each row of W_i^x is a scaled version of the associated row of W_i, and each element in b_i^x is a scaled version of b_i, i.e.,

$$W_i^x[p] = c^{(\phi(W_i x_i + b_i)[p])} \cdot W_i, \tag{8.5}$$

$$\text{and} \quad b_i^x[p] = c^{(\phi(W_i x_i + b_i)[p])} \cdot b_i. \tag{8.6}$$

For instance, if ReLU $\psi_{ReLU}(z) = \max(0, z)$ is used as the activation function $\psi(\cdot)$ at every layer i, we have an activation pattern $\phi(W_i x_i + b_i) \in \{0, 1\}^{d_i}$, where $\phi(W_i x_i + b_i)[p] = 0$ indicates that ReLU sets the output node p to 0 or otherwise preserves the node value. Therefore, at layer i, W_i^x and b_i^x are modified from W_i and b_i by setting the rows of W_i, whose corresponding activation patterns in $\phi(W_i x_i + b_i)$ are 0, to be all-zero vectors $\mathbf{0}$, and setting the associated elements in b_i^x to be 0 while other elements to be b_i.

We can apply the above process to deeper layers as well, eliminating all the ReLU functions to produce an x-specific local linear model representing one piece of the DNN, as shown in Eq. (8.3). Since the model is linear, the gradient $\frac{\partial f(x)}{\partial x}$ is the weight vector of the linear model. Also, given all the weights of the DNN, each linear region, and the associated linear model can be uniquely determined by the ReLU patterns $\{\phi(x_i)\}_{i=2}^m$, which are m binary vectors.

8.1.2 Attribution of DNN Outputs to Inputs

Given a specific input point x, the attribution of each dimension $f(x)[j]$ of the DNN output (e.g., the logit for class j) to the input features aims to assign a portion of $f(x)[j]$ to each of the input features i, and all the portions assigned to all the input features should sum up to $f(x)[j]$. For simplicity, in the rest of this chapter, we rename $f(x)$ to be $f(x)[j]$, which does not lose any generality since the same attribution method can be applied for any output dimension j. According to Eq. (8.3), $f(x)$ as a linear model on x can be decomposed into

two parts, the linear transformation $\frac{\partial f(x)}{\partial x}$ and the bias term b^x. The attribution of the first part is straightforward because we can directly assign each dimension of the gradient $\frac{\partial f(x)}{\partial x}$ to the associated input feature, and we can generate the gradient using the standard back-propagation algorithm. The gradient-based attribution methods have been widely studied in previous work (see Section 8.0.1). However, the attribution of the second part, i.e., the bias b, is arguably a more challenging problem since it is not obvious how to assign a portion of b to each input feature since b is a scalar value rather than a vector that, like the gradient, has the same dimensionality as the input vector.

One possible reason for the dearth of bias attribution studies might be that people consider bias, as a scalar, less important relative to the weight vector, containing only minor information about deep model decisions. The final bias scalar b^x of every local linear model, however, is the result of a complex process (see Eq. (8.3)), where the bias term on every neuron of a layer gets modified based on the activation function (e.g., for ReLU, a bias term gets dropped if the neuron has a negative value), then propagates to the next layer based on the weight matrix, and contribute to the patterns of activation function in the next layer. As the bias term applied to every neuron can be critical in determining the activation pattern (e.g., changing a neuron output from negative to positive for ReLU), we wish to be able to better understand the behavior of deep models by unveiling and reversing the process of how the final bias term is generated.

Moreover, as we show in our empirical studies (see Section 8.3), we train DNNs both with and without bias for image classification tasks, and the results show that the bias plays a significant role in producing accurate predictions. In fact, we find that it is not rare that the main component of a final logit, leading to the final predicted label, comes from the bias term, while the gradient term $\frac{\partial f(x)}{\partial x}x$ makes only a minor, or even negative, contribution to the ultimate decision. In such a case, ignoring the bias term can provide misleading input feature attributions.

Intuitively, the bias component also changes the geometric shape of the piecewise linear DNNs (see Fig. 8.1); this means that it is an essential component of deep models and should

also be studied, as we do in this work.

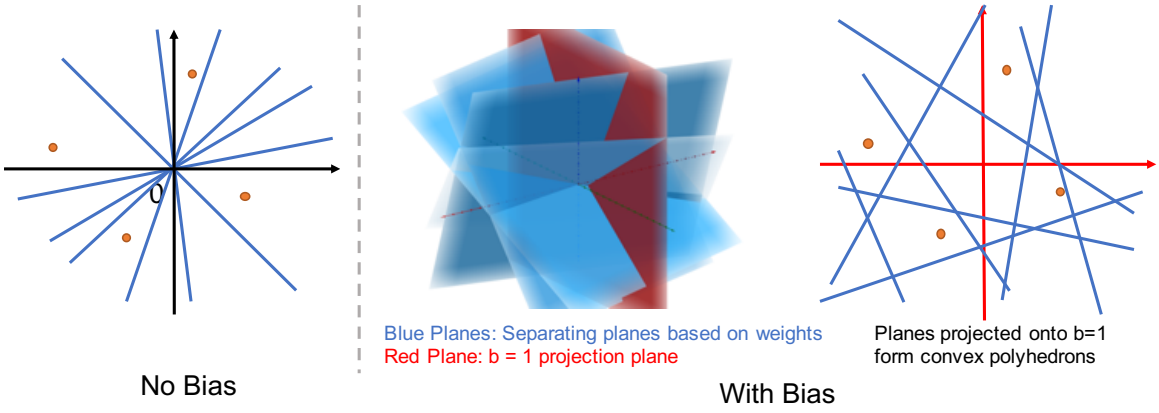


Figure 8.1: A Piecewise linear weight matrix divides the input plane into regions. Without the bias term, the regions are cones, while with the bias term, the regions are convex polyhedra.

It is a mathematical fact that a piecewise linear DNN is equivalent to a linear model for each input data point. Therefore, the interpretation of the DNN's behavior on the input data should be exclusive to the information embedded in the linear model. However, we often find that the gradient of the DNN, or the weight of the linear model, does not always produce satisfying explanations in practice, and in many cases, it may be due to the overlooked attribution of the bias term that contains the complementary or even key information to make the attribution complete.

8.2 Bias Backpropagation for Bias Attribution

In this section, we will introduce our method for bias attribution. In particular, the goal is to find a vector β of the same dimension d_{in} as the input data point x such that $\sum_{p=1}^{d_{in}} \beta[p] = b^x$. However, it is not clear how to directly assign a scalar value b to the d_{in} input dimensions, since there are m layers between the outputs and inputs. In the following, we explore the neural net structure for bias attribution and develop a backpropagation-type algorithm to

attribute the bias b layer by layer from the output $f(x)$ to the inputs in a bottom-up manner.

8.2.1 Bias Backpropagation (BBp)

Recall x_ℓ denotes the input nodes of layer $\ell \geq 2$, i.e.,

$$\begin{aligned} x_\ell =& \psi_{\ell-1}(W_{\ell-1}x_{\ell-1} + b_{\ell-1}) \\ =& \psi_{\ell-1}(W_{\ell-1}\psi_{\ell-2}(\dots\psi_1(W_1x + b_1)\dots) + b_{\ell-1}). \end{aligned} \tag{8.7}$$

According to the recursive computation shown in Eq. (8.3), the output $f(x)$ can be represented as a linear model of x_ℓ consisting of the gradient term and the bias term, as shown in the following lemma. Note that the bias term depends on the input x, as W_i^x and b_i^x are all modified based on input x.

Lemma 12. *Given x, the output $f(x)$ of a piecewise linear DNN can be written as a linear model of the input x_ℓ of any layer $\ell > 2$ ($x_1 = x$ is the raw input) in the following form.*

$$f(x) = \left(\prod_{i=\ell}^{m} W_i^x\right) x_\ell + \left(\sum_{j=\ell+1}^{m}\prod_{i=j}^{m} W_i^x b_{j-1}^x + b_m\right). \tag{8.8}$$

For each input node $x_\ell[p]$ of layer ℓ, we aim to compute $\beta_\ell[p]$ as the bias attribution on $x_\ell[p]$. We further require that summing $\beta_\ell[p]$ over all input nodes of layer ℓ recovers the bias in Eq. (8.8), i.e.,

$$\sum_{p=1}^{d_\ell} \beta_\ell[p] = \sum_{j=\ell+1}^{m}\prod_{i=j}^{m} W_i^x b_{j-1}^x + b_m, \tag{8.9}$$

so the linear model in Eq. (8.8) can be represented as the sum of d_ℓ terms associated with the d_ℓ input nodes, each composed of a linear transformation part and a bias attribution part, i.e.,

$$f(x) = \sum_{p=1}^{d_\ell}\left[\left(\prod_{i=\ell}^{m} W_i^x\right)[p]\cdot x_\ell[p] + \beta_\ell[p]\right]. \tag{8.10}$$

The above equation gives the attribution of the output $f(x)$ on each hidden node $x_\ell[p]$ of the DNN. It is composed of two parts, i.e., the gradient attribution and the bias attribution.

Since the bias in the right-hand side of Eq. (8.9) can be represented as an accumulated sum of the bias terms incurred from the last layer to layer ℓ (i.e., b_m for the last layer and $\prod_{i=j}^{m} W_i^x b_{j-1}^x$ for layer $j-1$), we can design a recursive rule that computes the bias attribution on layer $\ell-1$ given the bias attribution β_ℓ on layer ℓ. In particular, we assign different portions of $\beta_\ell[p]$ to each node $x_{\ell-1}[q]$ on layer $\ell-1$, and make sure that summing up those portions recovers $\beta_\ell[p]$. Each portion $B_\ell[p,q]$ can be treated as a message regarding bias attribution that node $x_\ell[p]$ sends to node $x_{\ell-1}[q]$. For each node $x_\ell[p]$ of layer ℓ, we compute a vector of attribution scores $\alpha_\ell[p]$, and define the message $B_\ell[p,q]$ as

$$B_\ell[p,q] \triangleq \alpha_\ell[p,q] \times \beta_\ell[p], \tag{8.11}$$

$$\sum_{q=1}^{d_{\ell-1}} \alpha_\ell[p,q] = 1 \quad \text{and,} \quad \forall p \in [d_\ell], q \in [d_{\ell-1}]. \tag{8.12}$$

We will discuss several options to compute the attribution scores $\alpha_\ell[p]$ later. To make our bias attribution method flexible and compatible with any attribution function, we allow both negative scores and positive scores in $\alpha_\ell[p]$.

The bias attribution $\beta_{\ell-1}[q]$ on node $x_{\ell-1}[q]$ of layer $\ell-1$ is achieved by firstly summing up the bias attribution messages sent from nodes in layer ℓ, and then adding the bias term $\prod_{i=\ell}^{m} W_i^x b_{j-1}^x$ incurred in layer $\ell-1$ (which is applied to all nodes in layer $\ell-1$), as shown below:

$$\beta_{\ell-1}[q] = \prod_{i=\ell}^{m} W_i^x b_{j-1}^x + \sum_{p=1}^{d_\ell} B_\ell[p,q]. \tag{8.13}$$

It can be easily verified that summing up the attribution $\beta_{\ell-1}[q]$ over all the nodes on layer $\ell-1$ yields the bias term in Lemma 12, when writing $f(x)$ at x as a linear model of $x_{\ell-1}$, i.e.,

$$\sum_{q=1}^{d_{\ell-1}} \beta_{\ell-1}[q] = \sum_{j=\ell}^{m} \prod_{i=j}^{m} W_i^x b_{j-1}^x + b_m. \tag{8.14}$$

Hence, the complete attribution of $f(x)$ on the nodes of layer $\ell-1$ can be written in the

same form as the one shown in Eq. (8.10) for layer ℓ, i.e.,

$$f(x) = \sum_{q=1}^{d_{\ell-1}} \left[\left(\prod_{i=\ell-1}^{m} W_i^x \right) [q] \cdot x_{\ell-1}[q] + \beta_{\ell-1}[q] \right]$$

. Therefore, we start from the last layer, and recursively apply Eq. (8.11)-(8.13) from the last layer to the first layer. This process backpropagates to the lower layers the bias term incurred in each layer and the bias attributions sent from higher layers. Eventually, we can obtain the bias attribution $\beta[p]$ for each input dimension p. The bias attribution algorithm is detailed in Algorithm 14.

Algorithm 14: Bias Backpropagation (BBp)

input : x, $\{W_\ell\}_{\ell=1}^m$, $\{b_\ell\}_{\ell=1}^m$, $\{\psi_\ell(\cdot)\}_{\ell=1}^m$

Compute $\{W_\ell^x\}_{\ell=1}^m$ and $\{b_\ell^x\}_{\ell=1}^m$ for x by Eq. (8.5) ; `// Get data point specific weight/bias`

$\beta_m \leftarrow b_m$; `//` β_ℓ `holds the accumulated attribution for layer` ℓ

for $\ell \leftarrow m$ **to** 2 **by** -1 **do**

  for $p \leftarrow 1$ **to** d_ℓ **by** 1 **do**

    Compute $\alpha_\ell[p]$ by Eq. (8.15)-(8.17) or Eq. (8.18) ; `// Compute attribution score`

    $B_\ell[p, q] \leftarrow \alpha_\ell[p, q] \times \beta_\ell[p], \ \ \forall\, q \in [d_{\ell-1}]$; `// Attribute to the layer input`

  end

  for $q \leftarrow 1$ **to** $d_{\ell-1}$ **by** 1 **do**

    $\beta_{\ell-1}[q] \leftarrow \prod_{i=\ell}^{m} W_i^x b_{j-1}^x + \sum_{p=1}^{d_\ell} B_\ell[p, q]$; `// Combine with bias of layer` $\ell - 1$

  end

end

return $\beta_1 \in \mathbb{R}^{d_{in}}$;

8.2.2 Options to Compute Attribution Scores in $\alpha_\ell[p]$

In the following, we discuss three possible options to compute the attribution scores in $\alpha_\ell[p]$, where $\alpha_\ell[p,q]$ measures how much of the bias $x_\ell[p]$ should be attributed to $x_{\ell-1}[q]$. For the first option, we design $\alpha_\ell[p]$ so that the bias attribution on each neuron serves as a *compensation* for the weight or gradient term to achieve the desired output value, and for the other two options, we design $\alpha_\ell[p]$ based on the *contribution* of the gradient term.

We have $x_\ell[p] = \sum_{r:=1}^{d_{l-1}} W^x_{\ell-1}[p,r]x_{\ell-1}[r] + b^x_\ell[p]$. Suppose $b^x_\ell[p]$ is negative, we may reason that to achieve the target value of $x_\ell[p]$, the positive components of the gradient term $\sum_{r:=1}^{d_{l-1}} W^x_{\ell-1}[p,r]x_{\ell-1}[r]$ are larger than desirable, so that we need to apply the additional negative bias in order to achieve the desired output $x_\ell[p]$. In other words, the large positive components can be thought as the causal factor leading to the negative bias term, so we attribute more bias to the larger positive components.

On the other hand, suppose $b^x_\ell[p]$ is positive, then the negative components of the gradient term are smaller (or larger in magnitude) than desirable, so the small negative values cause the bias term to be positive, and therefore, we attribute more bias to the smaller negative components. Thus, we have

$$\alpha_\ell[p,q] = \frac{\mathbb{1}_{e(l-1,p,q)=1}\exp(s_\ell[p,q]/T)}{\sum_{r=1}^{d_{\ell-1}}\mathbb{1}_{e(l-1,p,r)=1}\exp(s_\ell[p,r]/T)}, \tag{8.15}$$

$$\text{where } s_\ell[p,q] = -\operatorname{sign}(b^x_\ell[p])\cdot W^x_{\ell-1}[p,q]x_{\ell-1}[q], \tag{8.16}$$

$$e(l-1,p,q) = |\operatorname{sign}(W^x_{\ell-1}[p,q]x_{\ell-1}[q])|. \tag{8.17}$$

We use the logistic function to attribute the bias so that the sum over all components recovers the original bias, and T serves as a temperature parameter to control the sharpness of the attribution. With T large, the bias is attributed in a more balanced manner, while with T small, the bias is attributed mostly to a few neurons in layer $\ell-1$. Also note that we only consider the non-zero components (indicator $\mathbb{1}_{e(l-1,p,q)=1}$ checks whether the component is zero), as the zero-valued components do not offer any contribution to the output value. For

example, consider a convolutional layer, the corresponding matrix form is very sparse, and only the non-zero entries are involved in the convolution computation with a filter.

The second option adopts a different philosophy of attributing based on the *contribution* of the gradient term. Again, the target is to achieve the value of $x_\ell[p]$, and we may assume that to achieve such a value, every component $W^x_{\ell-1}[p,r]x_{\ell-1}[r]$ should have an equal responsibility, which is the average target value, i.e., $x_\ell[p]/\sum_{r=1}^{d_{l-1}} \mathbb{1}_{e(l-1,p,q)=1}$ (again, we only need to consider the contribution from non-zero components). The offsets of each component to the average target value can be treated as the contribution of each feature to the output of the layer, and we attribute the bias term based on the exponentiated values of the contribution. This produces the following method to compute $s_\ell[p,q]$, i.e.,

$$s_\ell[p,q] = \frac{x_\ell[p]}{\sum_{r=1}^{d_{l-1}} \mathbb{1}_{e(l-1,p,q)=1}} - W^x_{\ell-1}[p,q]x_{\ell-1}[q]. \tag{8.18}$$

Note we use the same equations for $e(l-1,p,q)$ and $\alpha_l[p,q]$ as defined in Eq. (8.15)-(8.17)

The third option utilizes a similar idea of attributing based on contribution of the gradient term, but it is specific for the ReLU nonlinearity. Since hidden neurons are non-negative for ReLU networks, we may consider only the positive part of the gradient term as the contribution. Thus, we propose the following option for the attribution score:

$$\alpha_\ell[p,q] = \frac{\mathbb{1}_{e^+(l-1,p,q)=1} W^x_{\ell-1}[p,q]x_{\ell-1}[q]}{\sum_{r=1}^{d_{\ell-1}} \mathbb{1}_{e^+(l-1,p,r)=1} W^x_{\ell-1}[p,r]x_{\ell-1}[r]}, \tag{8.19}$$

$$\text{where } e^+(l-1,p,q) = \text{sign}(W^x_{\ell-1}[p,q]x_{\ell-1}[q]). \tag{8.20}$$

The above options are our designs for the $\alpha_\ell[p,q]$ function. The attribution function is valid as long as $\sum_{r=1}^{d_{l-1}} \alpha_\ell[p,r] = 1$. While for the first two options, we utilize the logistic function so that the attribution factors are positive, $\alpha_\ell[p,r]$ can be negative and still applicable to our BBp framework. We note that there is no single solution to get the optimal attribution function. The first two proposed options can be applied to any piecewise-linear deep neural networks, and for specific activation function, it is possible to design specialized attribution functions to get still better bias attribution (like the third option).

8.3 Experiments

8.3.1 Importance of Bias in DNNs

We first evaluate the importance of bias terms, or in other words, the amount of information encoded in the bias terms by comparing networks trained both with and without bias.

In Table 8.1, we compare results on the CIFAR-10, CIFAR-100 [68] and Fashion MNIST [144] datasets. We trained using the VGG-11 Network of [112], and we compare the results trained with bias, and without bias. Moreover, in the trained with bias case, we derive the linear model of every data point $g(x) = wx + b$, and compare the performance using only the resulting gradient term wx and only the resulting bias term b. From the results shown in the table, we find that the bias term carries appreciable information and makes unignorable contributions to the correct prediction of the network.

Table 8.1: Compare the performance (in test accuracy %) of models with/without the bias terms. The "only wx" and "only b"columns use the same model as the "train with bias" column.

Dataset	Train Without Bias	Train With Bias, Test All	Test Only wx	Test Only b
CIFAR10	87.0	90.9	71.5	62.2
CIFAR100	62.8	66.8	40.3	36.5
FMNIST	94.1	94.7	76.1	24.6

8.3.2 Bias Attribution Analysis Visualization

We present our bias attribution results, using the three options of attribution scores discussed in section 8.2.2, and compare to the attribution result based only on gradient information. We test BBp on STL-10 [23] and ImageNet(ILSVRC2012) [102] and show the results in Fig. 8.4. For STL-10, we use a 10-layer convolutional network ((32,3,1), maxpool, (64,3,1), maxpool, (64,3,1), maxpool, (128,3,1), maxpool, (128,3,1), and dense10, where (i,j,k) cor-

responds to a convolutional layer with i channels, kernel size j and padding k), and for ImageNet we use the VGG-11 network of [112]. For both gradient and bias attribution, we select the gradient/bias corresponding to the predicted class (i.e., one row for the final layer weight matrix and one scalar for the final layer bias vector). Note that BBp is a general framework and can work with other choices of gradients and biases for the last layer (e.g. the top predicted class minus the second predicted class).

From Fig. 8.4, we present visualizations of bias attribution compared to gradient attribution and integrated gradient [120] attribution (50 steps approximation, reference image all black) on ImageNet and STL-10 datasets. The label of every image is shown in the left-most column. The gradient attribution is the element-wise product between the linear model weight w and data point x. The "norm.grad.", "norm.integrad." and "norm.bias." columns show the attribution of gradient, integrated gradient and bias normalized to the color range (0-255). The "grad.attrib", "integrad.attrib" and "bias.attrib" show the 10% data features with the highest attribution magnitude of the original image. Bias1 correspond to the first proposed option of calculating the bias attribution score (Eq. (8.15)-(8.17)), bias2 (Eq. (8.18)) corresponds to the second proposed option and bias3(Eq. (8.19)-(8.20)) corresponds to the third. For all options of calculating the bias attribution score, the temperature parameter T is set to 1. We can observe that the bias attribution can highlight meaningful features in the input data, and in many cases, capture the information that is complementary to the information provided by the gradient. For example, for the "Brambling" image from ImageNet, BBp shows stronger attribution on the bird's head and wings compared to the gradient method. For the "Fire- guard" image of ImageNet, BBp has clear attribution to the fire, in addition to the shape of the guard, while the gradient method only shows the shape of the guard. Similarly, for the "folding chair" of ImageNet, BBp shows clearer parts of the chair, while the gradient attribution shows less relevant features such as the background wall. Statistically, on ImageNet dataset, 59.4%, 56.3% and 55.2% of the 3 bias attribution pixels (top 10% response) are not included in the gradient attribution, and on STL-10 dataset, the portions are 50.52%, 48.29%, and 43.94% More visualizations can be found the appendix.

8.3.3 Bias Attribution for Various Layers

As we can naturally decompose the overall bias term into biases of individual layers, our BBp method has the advantage of investigating the attributions of biases of various layers of a given network. From Fig. 8.2, we compare attributions of three options of BBp on ImageNet dataset with biases on various layers of the vgg-11 network. To exclude the biases of certain layers, we run BBp with the corresponding biases set to zeros. We can observe that attribution from all layers tend to give the most complete shape of the objects. As we exclude layers from the input, option 1 gives attributions more concentrated on parts of the objects, such as the head parts of the dog (the 2nd image) and the bird (the 3rd image), while options 2 and 3 focus more on the contours of the objects.

8.3.4 MNIST Digit Flip Test

This experiment was proposed in [110] to verify that the attribution method is class sensitive. A digit image of MNIST [74] dataset gets modified by a mask based on its difference of attributions from two different classes, so that the image should have fewer important features for the source class, and more important features for the target class. Then we measure the neural network's output on the modified image to check if the prediction shifts from the source class to the target class (the log-odds score is $(f(x)[c_1] - f(x)[c_2]) - (f(\hat{x})[c_1] - f(\hat{x})[c_2])$, where f is the network, x is an image, $\hat{x}$ is the modified x based on the attribution and c_1, c_2 correspond to the source and target class). From Fig. 8.3, we see that the bias attribution methods are class sensitive and comparable to methods such as integrated gradient and DeepLift.

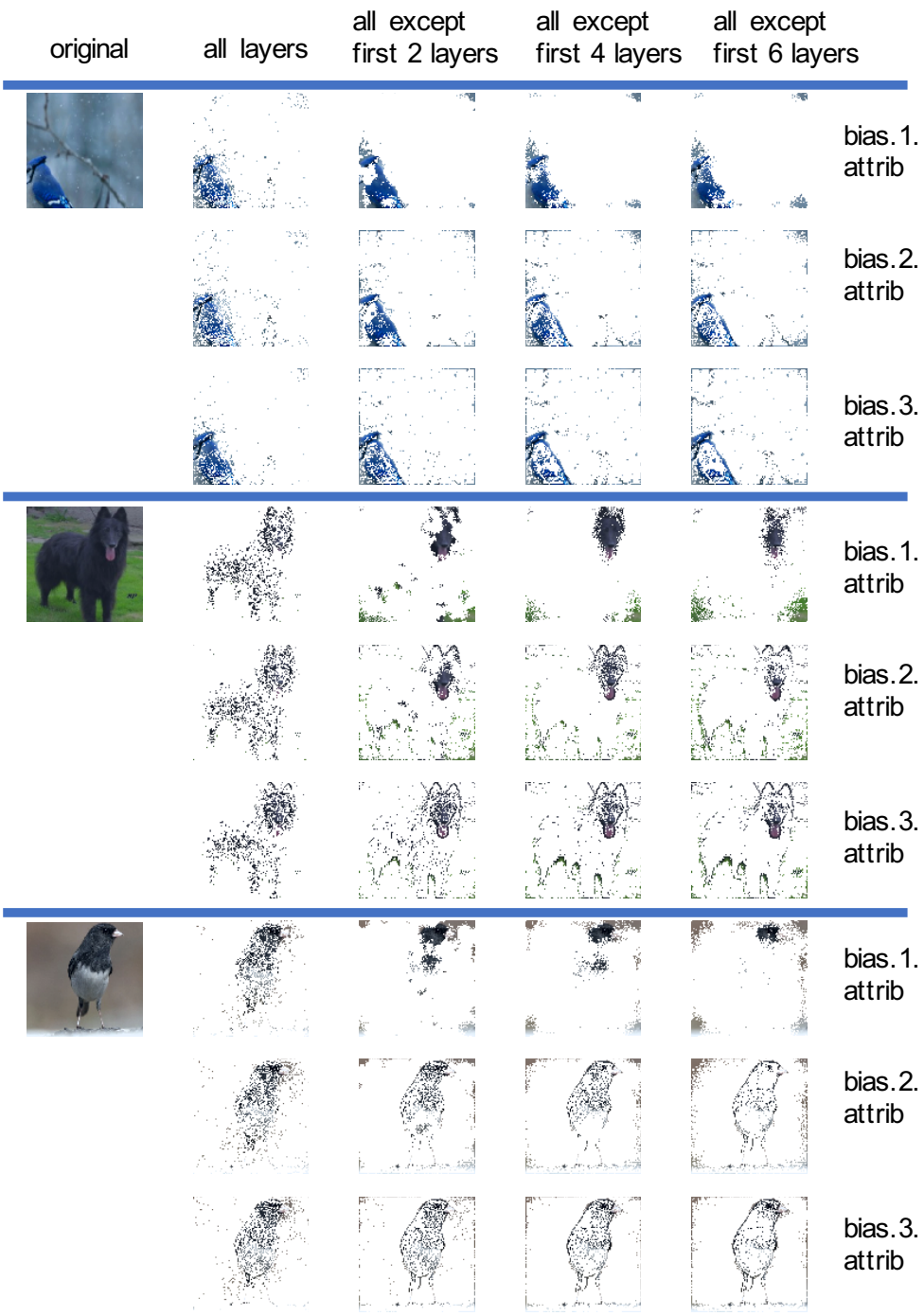


Figure 8.2: Bias attribution on ImageNet with biases on different layers of the vgg-11 network. "bias.1(2,3)" corresponds to the three attribution score options proposed in section 8.2.2

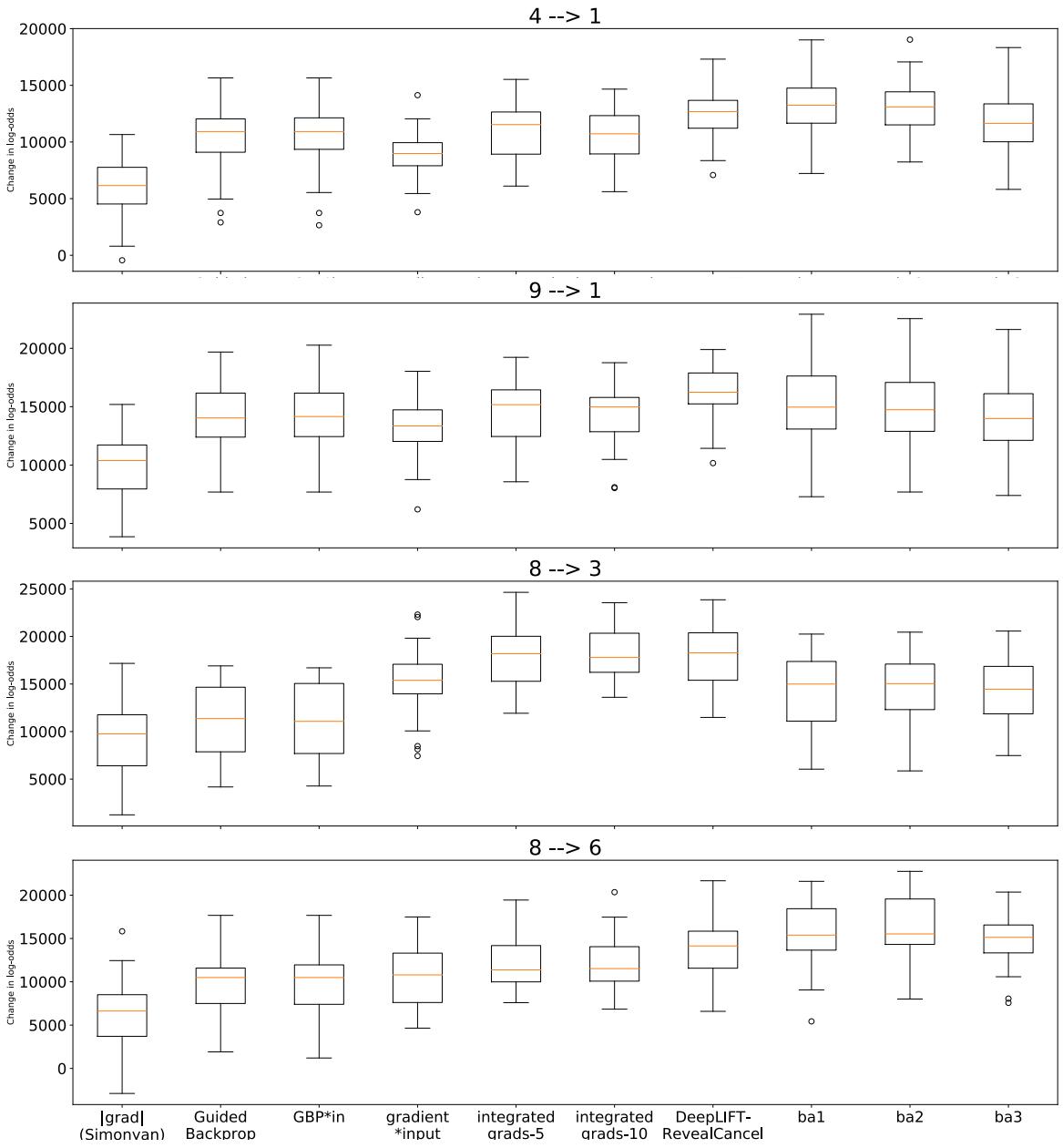


Figure 8.3: MNIST digit flip test: boxplots of increase in log-odds scores of target vs. source class after the features removed. "Integrated grads-n" refers to the integrated gradient method with n step approximations. "ba1, ba2 and ba3" refer to our 3 options of bias attribution.

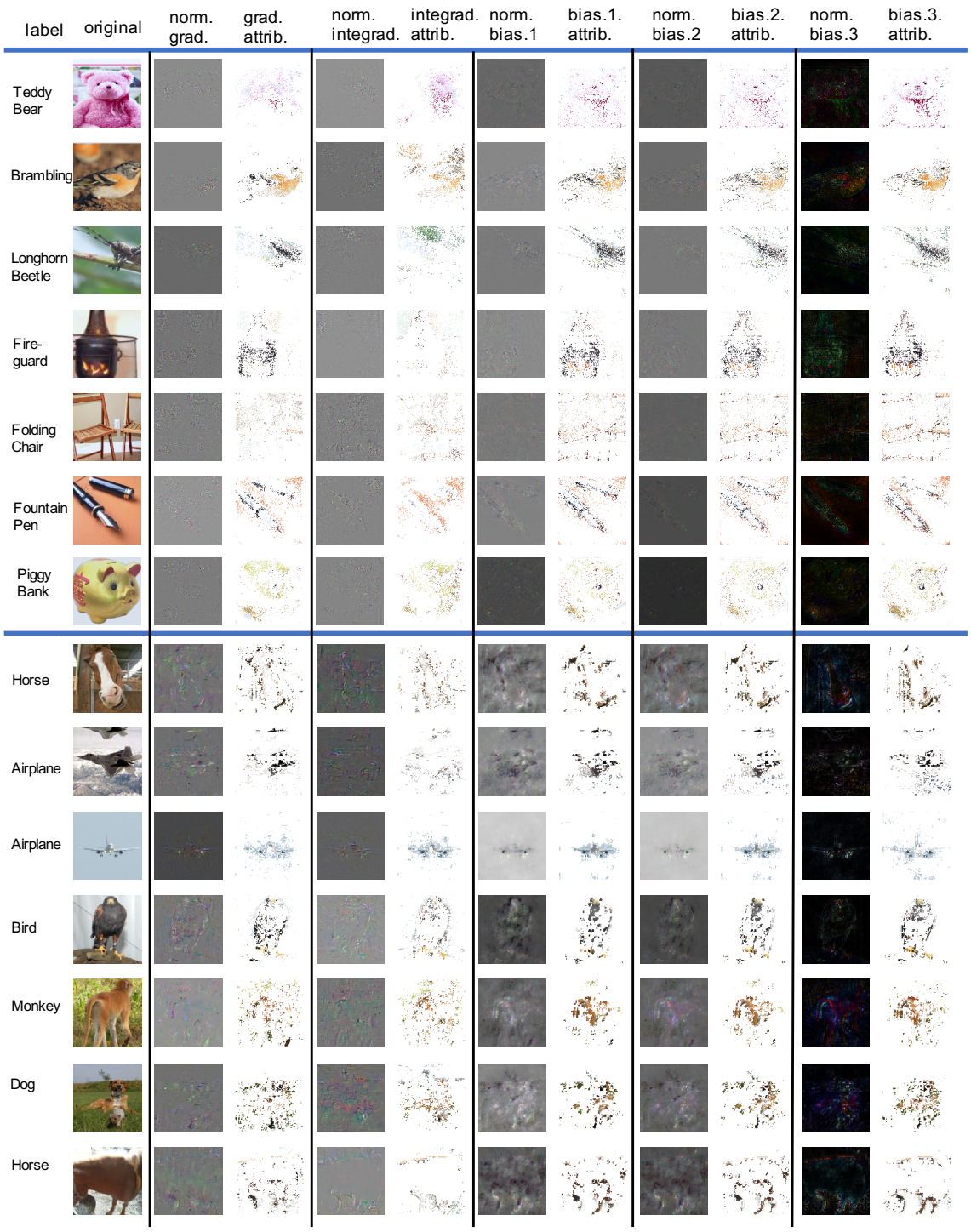


Figure 8.4: Bias attribution on the ImageNet (top) and STL-10 (bottom) datasets compared to gradient and integrated gradient attribution.

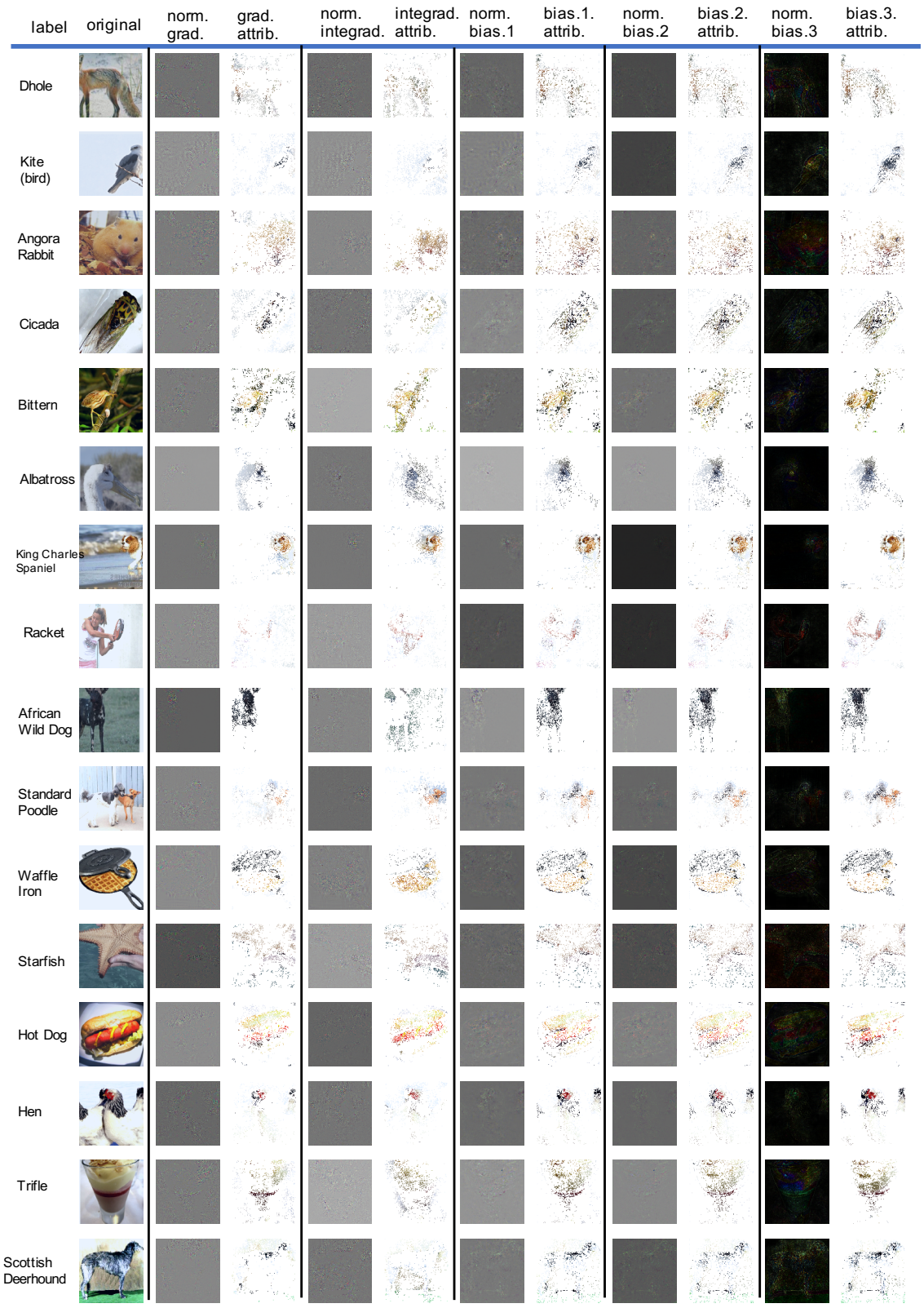


Figure 8.5: More visualizations of bias attribution on the ImageNet compared to gradient and integrated gradient attribution.

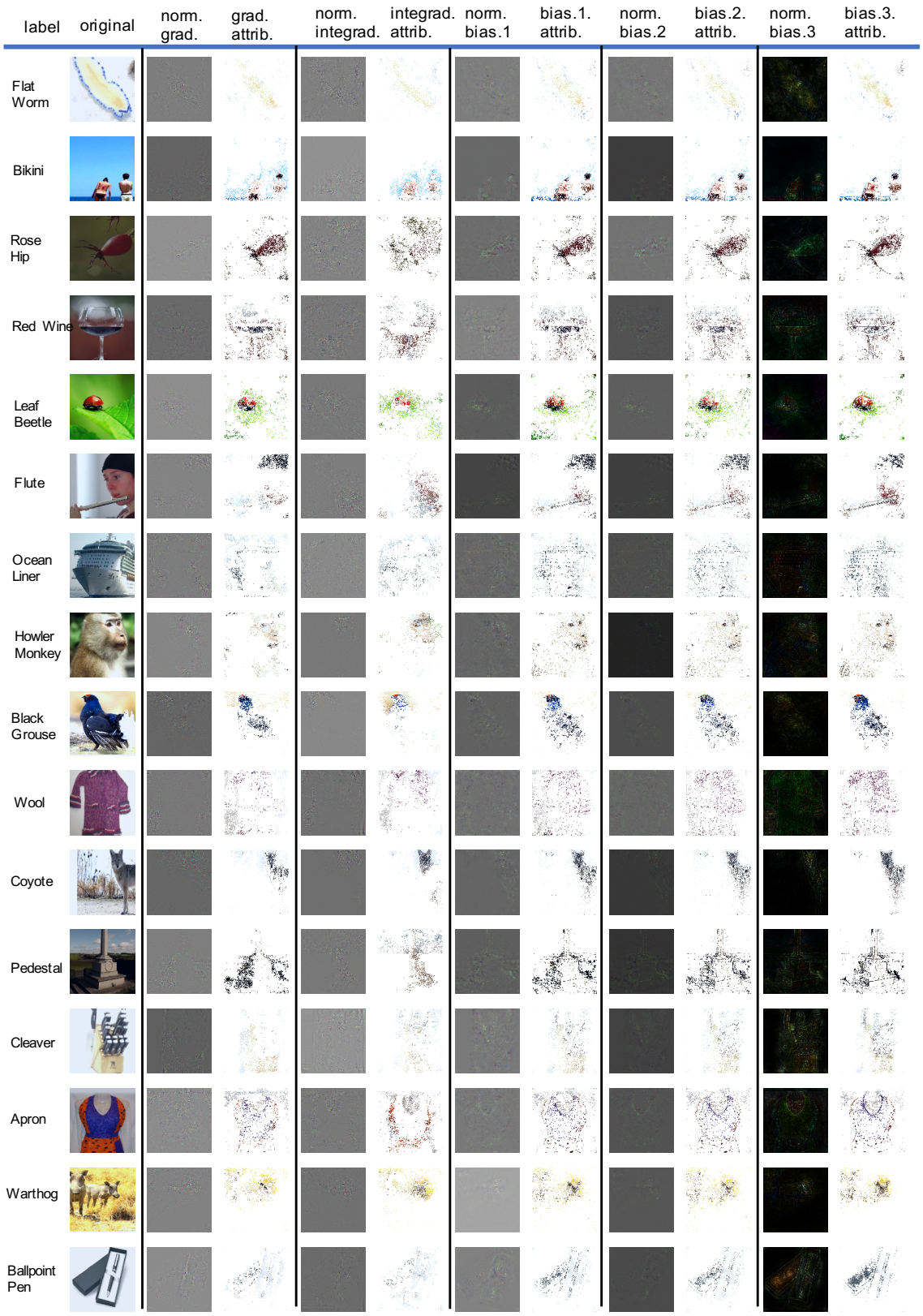


Figure 8.6: More visualizations of bias attribution on the ImageNet compared to gradient and integrated gradient attribution.

Chapter 9

CONCLUSIONS

The first part of this dissertation focuses on the problem of robust submodular partitioning. Particularly, we propose algorithms with theoretical guarantees for the robust submodular partitioning problem under the unconstrained and the constrained settings. For the constrained setting, we investigate the cardinality constrained case and apply it to the task of generating deterministic and high-quality mini-batches for stochastic gradient methods with significantly improved performance in practice compared to randomly generated mini-batches. The robust submodular partitioning problem has potentially many other practical applications, such as distributing training data onto multiple computation nodes for parallel training. There are also many open problems for future research: 1) we focus on the partitioning problem in the homogeneous setting where the same submodular function is applied for each block in the partition. A more general problem, often referred to as the heterogeneous setting, allows different submodular functions for different partition blocks. This is a more general setting, and the current techniques presented in the dissertation do not immediately apply. We think a more feasible approach to solve the heterogeneous problem is to address two easier versions of the problem first. For both versions, we still have the same submodular function for every block. For the first version, we have a combined constraint on all blocks (e.g., all blocks share a matroid constraint, as opposed to the current case where every block has the same matroid constraint), and for the second version, we have a different constraint for each block in the partition. It can be shown that the heterogeneous problem generalizes the two versions, and we can use some of the current techniques for the two versions as we still have the same function for each block.

The second part of the dissertation discusses a line of research about linear models of

ReLU deep neural networks, covering three works that analyze, improve and explain the ReLU deep network. Based on the intuition that a ReLU deep network is essentially a linear model for every data point, we utilize the linear models encoded in the ReLU network to analyze the complexity of the network by analyzing the spectrum of the Extended Data Jacobian Matrix. Next, we propose an improved version of dropout, called jumpout, that enforces the linear models over different data points to be locally smooth. Finally, we show an interpretation method that focuses on attributing the bias term in the linear model to the input features to explain the network's behavior. Even though the methods are applied to ReLU deep networks, they can be easily adapted to more general cases for networks with piecewise linear activation functions. Even though deep learning has achieved great success, we still do not understand deep models very well. The linear model view of the deep neural networks offers a different angle to understand the network behavior geometrically. For future work, it is worth investigating how do the linear models evolve over the training process to understand the convergence and generalization performance of the network.

BIBLIOGRAPHY

[1] Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[2] Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *SICOMP*, 2010.

[3] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[4] Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

[5] Lei Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *NeurIPS*, pages 3084–3092, 2013.

[6] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, 2014.

[7] Ashwinkumar Badanidiyuru, Amin Karbasi, Ehsan Kazemi, and Jan Vondrák. Submodular maximization through barrier functions. *arXiv preprint arXiv:2002.03523*, 2020.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Siddharth Barman and Sanath Kumar Krishna Murthy. Approximation algorithms for

maximin fair division. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 647–664, 2017.

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[11] Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. In *International Conference on Artificial Neural Networks*, pages 63–71. Springer, 2016.

[12] S.P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge Univ Pr, 2004.

[13] Niv Buchbinder and Moran Feldman. Deterministic algorithms for submodular maximization problems. *ACM Transactions on Algorithms (TALG)*, 14(3):1–20, 2018.

[14] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *FOCS*, 2012.

[15] Niv Buchbinder, Moran Feldman, Joseph Seffi Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1433–1452. SIAM, 2014.

[16] G. Calinescu, C. Chekuri, M. Pal, and J. Vondrák. Maximizing a monotone submodular function under a matroid constraint. *IPCO*, 2007.

[17] Deeparnab Chakrabarty, Prateek Jain, and Pravesh Kothari. Provable submodular minimization using wolfe's algorithm. In *Advances in Neural Information Processing Systems*, pages 802–809, 2014.

[18] C. Chekuri, J. Vondrák, and R. Zenklusen. Submodular function maximization via

the multilinear relaxation and contention resolution schemes. *Proceedings of the 43rd annual ACM symposium on Theory of computing*, 2011.

[19] Chandra Chekuri and Alina Ene. Approximation algorithms for submodular multiway partition. In *FOCS*, 2011.

[20] Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 575–584. IEEE, 2010.

[21] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[22] Yuxin Chen and Andreas Krause. Near-optimal batch mode active learning and adaptive submodular optimization. In *International Conference on Machine Learning*, pages 160–168, 2013.

[23] Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, pages 215–223, 2011.

[24] Andrew Cotter, Mahdi Milani Fard, Seungil You, Maya Gupta, and Jeff Bilmes. Constrained interacting submodular groupings. In *International Conference on Machine Learning*, pages 1068–1077. PMLR, 2018.

[25] Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury. Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(9):1469–1477, 2015.

[26] William H Cunningham. Testing membership in matroid polyhedra. *Journal of Combinatorial Theory, Series B*, 36(2):161–188, 1984.

[27] A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *arXiv preprint arXiv:1102.3975*, 2011.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[29] J. Edmonds. Submodular functions, matroids and certain polyhedra. *Combinatorial structures and their Applications*, 1970.

[30] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[31] Uriel Feige, Vahab Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. COMPUT.*, 40(4):1133–1155, 2011.

[32] M. Feldman, J.S. Naor, and R. Schwartz. A unified continuous greedy algorithm for submodular maximization. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 570–579. IEEE, 2011.

[33] M.L. Fisher, G.L. Nemhauser, and L.A. Wolsey. An analysis of approximations for maximizing submodular set functions—II. In *Polyhedral combinatorics*, 1978.

[34] Satoru Fujishige. *Submodular functions and optimization*, volume 58. Elsevier, 2005.

[35] Satoru Fujishige, Takumi Hayashi, and Shigueo Isotani. *The minimum-norm-point algorithm applied to submodular function minimization and linear programming*. Citeseer, 2006.

[36] Victor Gabillon, Branislav Kveton, Zheng Wen, Brian Eriksson, and S Muthukrishnan. Adaptive submodular maximization in bandit setting. In *NIPS*, 2013.

[37] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1098–1116. SIAM, 2011.

[38] Mohammad Ghodsi, MohammadTaghi HajiAghayi, Masoud Seddighin, Saeed Seddighin, and Hadi Yami. Fair allocation of indivisible goods: Improvements and generalizations. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 539–556, 2018.

[39] Daniel Golovin. Max-min fair allocation of indivisible goods. *Technical Report CMU-CS-05-144*, 2005.

[40] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[45] GE Hinton, Oriol Vinyals, and Jeff Dean. Dark knowledge. *Presented as the keynote in BayLearn*, 2014.

[46] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[47] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[48] Steven CH Hoi, Rong Jin, Jianke Zhu, and Michael R Lyu. Batch mode active learning and its application to medical image classification. In *ICML*, 2006.

[49] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV*, pages 646–661, 2016.

[50] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[51] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[52] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001.

[53] R. Iyer and J. Bilmes. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints. In *NIPS*, 2013.

[54] R. Iyer, S. Jegelka, and J. Bilmes. Fast semidifferential based submodular function optimization. In *ICML*, 2013.

[55] Rishabh Krishnan Iyer. *Submodular optimization and machine learning: Theoretical results, unifying and scalable algorithms, and applications*. PhD thesis, 2015.

[56] S. Jegelka and J. Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *CVPR*, 2011.

[57] Yangqing Jia. Tutorial on training imagenet with caffe. `http://caffe.berkeleyvision.org/gathered/examples/imagenet.html`, 2014.

[58] Haotian Jiang. Minimizing convex functions with integral minimizers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 976–985. SIAM, 2021.

[59] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems*, pages 2078–2086, 2014.

[60] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proc. SIGKDD*, pages 137–146. ACM, 2003.

[61] Subhash Khot and Ashok Ponnuswami. Approximation algorithms for the max-min allocation problem. In *APPROX*, 2007.

[62] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[63] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NeurIPS*, pages 2575–2583. 2015.

[64] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[65] A. Krause and C. Guestrin. A note on the budgeted maximization of submodular

functions, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.3314.

[66] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. In *JMLR*, 2008.

[67] Andreas Krause, Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Robust submodular observation selection. In *JMLR*, 2008.

[68] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[69] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[71] Alan Kuhnle. Interlaced greedy algorithm for maximization of submodular functions in nearly linear time. *arXiv preprint arXiv:1902.06179*, 2019.

[72] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 545–554. Society for Industrial and Applied Mathematics, 2009.

[73] Chandrashekhar Lavania, Kai Wei, Rishabh Iyer, and Jeff Bilmes. A practical online framework for extracting running video summaries under a fixed memory budget. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 226–234. SIAM, 2021.

[74] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[75] J. Lee, V.S. Mirrokni, V. Nagarajan, and M. Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *STOC*, pages 323–332. ACM, 2009.

[76] H. Lin and J. Bilmes. Learning mixtures of submodular shells with application to document summarization. In *Uncertainty in Artificial Intelligence*, 2012.

[77] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL/HLT-2011)*, Portland, OR, June 2011.

[78] Yuzong Liu, Kai Wei, Katrin Kirchhoff, Yisong Song, and Jeff Bilmes. Submodular feature selection for high-dimensional acoustic score spaces. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Vancouver, Canada, 2013.

[79] L. Lovász. Submodular functions and convexity. *Mathematical Programming*, 1983.

[80] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, 1978.

[81] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. Lazier than lazy greedy. In *Proc. AAAI*, 2015.

[82] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *AAAI*, pages 1812–1818, 2015.

[83] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[84] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, volume 70, pages 2498–2507, 2017.

[85] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[86] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *NeurIPS*, pages 2924–2932. 2014.

[87] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[88] K. Nagano, Y. Kawahara, and K. Aihara. Size-constrained submodular minimization through minimum norm base. In *ICML*, 2011.

[89] Kiyohito Nagano, Yoshinobu Kawahara, and Satoru Iwata. Minimum average cost clustering. In *Advances in Neural Information Processing Systems*, pages 1759–1767, 2010.

[90] Martin Nägele, Benny Sudakov, and Rico Zenklusen. Submodular minimization under congruency constraints. *Combinatorica*, 39(6):1351–1386, 2019.

[91] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

[92] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[93] Behnam Neyshabur, Ruslan R Salakhutdinov, and Nati Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2413–2421. Curran Associates, Inc., 2015.

[94] J.B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 2009.

[95] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

[96] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[97] Adarsh Prasad, Stefanie Jegelka, and Dhruv Batra. Submodular meets structured: Finding diverse subsets in exponentially-large structured item sets. In *NIPS*, 2014.

[98] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2847–2854, 2017.

[99] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, 2016.

[100] Colorado Reed and Zoubin Ghahramani. Scaling the indian buffet process via submodular maximization. *arXiv preprint arXiv:1304.3285*, 2013.

[101] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[102] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[103] Farnood Salehi, Elisa Celis, and Patrick Thiran. Stochastic optimization with bandit sampling. *arXiv preprint arXiv:1708.02544*, 2017.

[104] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[105] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

[106] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra, et al. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626, 2017.

[107] Manohar Shamaiah, Siddhartha Banerjee, and Haris Vikalo. Greedy sensor selection: Leveraging submodularity. In *Proc. CDC*, pages 2572–2577. IEEE, 2010.

[108] Ohad Shamir. Without-replacement sampling for stochastic gradient methods. In *Advances in Neural Information Processing Systems*, pages 46–54, 2016.

[109] Yusuke Shinohara. A submodular optimization approach to sentence set selection. In *ICASSP*, pages 4112–4115. IEEE, 2014.

[110] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017.

[111] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[112] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[113] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[114] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[115] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. In *NeurIPS*, pages 28–36. 2016.

[116] Adish Singla, Ilija Bogunovic, Gábor Bartók, Amin Karbasi, and Andreas Krause. Near-optimally teaching the crowd to classify. *arXiv preprint arXiv:1402.2092*, 2014.

[117] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[118] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[119] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[120] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

[121] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[122] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[123] Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. In *FOCS*, 2008.

[124] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[125] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[126] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. In *JMLR*, 2005.

[127] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems*, pages 1413–1421, 2014.

[128] Stefan Uhlich, Marcello Porcu, Franck Giron, Michael Enenkl, Thomas Kemp, Naoya Takahashi, and Yuki Mitsufuji. Improving music source separation based on deep neural networks through data augmentation and network blending. In *Acoustics, Speech*

and Signal Processing (ICASSP), 2017 IEEE International Conference on, pages 261–265. IEEE, 2017.

[129] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *ArXiv*, abs/1607.08022, 2016.

[130] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, 2008.

[131] Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

[132] Shengjie Wang, Abdel-rahman Mohamed, Rich Caruana, Jeff Bilmes, Matthai Plilipose, Matthew Richardson, Krzysztof Geras, Gregor Urban, and Ozlem Aslan. Analysis of deep neural networks with extended data jacobian matrix. In *International Conference on Machine Learning*, pages 718–726, 2016.

[133] Shengjie Wang, Wenruo Bai, Chandrashekhar Lavania, and Jeff Bilmes. Fixing mini-batch sequences with hierarchical robust partitioning. volume 89 of *Proceedings of Machine Learning Research*, pages 3352–3361, 2019.

[134] Shengjie Wang, Tianyi Zhou, and Jeff A. Bilmes. Bias also matters: Bias attribution for deep neural network explanation. In *ICML*. 2019.

[135] Sida Wang and Christopher Manning. Fast dropout training. In *ICML*, volume 28, pages 118–126, 2013.

[136] Kai Wei. *Submodular Optimization and Data Processing*. PhD thesis, 2016.

[137] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *North American Chapter of the Association for Computational Linguistics/Human Language Technology Conference (NAACL/HLT-2013)*, Atlanta, GA, June 2013.

[138] Kai Wei, Yuzong Liu, Katrin Kirchhoff, Chris Bartels, and Jeff Bilmes. Submodular subset selection for large-scale speech training data. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Florence, Italy, 2014.

[139] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *ICML*, 2015.

[140] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *ICML*, 2015.

[141] Kai Wei, Rishabh K Iyer, Shengjie Wang, Wenruo Bai, and Jeff A Bilmes. Mixed robust/average submodular partitioning: Fast algorithms, guarantees, and applications. In *Advances in Neural Information Processing Systems*, pages 2233–2241, 2015.

[142] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[143] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[144] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[145] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):23, 2012.

[146] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

[147] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[148] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[149] Ke Zhai and Huan Wang. Adaptive dropout with rademacher complexity regularization. In *ICLR*, 2018.

[150] Cheng Zhang, Hedvig Kjellström, and Stephan Mandt. Determinantal point processes for mini-batch diversification. In *33rd Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, 11 August 2017 through 15 August 2017*. AUAI Press Corvallis, 2017.

[151] Jingjing Zheng, Zhuolin Jiang, Rama Chellappa, and Jonathon P Phillips. Submodular attribute selection for action recognition in video. In *NIPS*, 2014.

[152] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, 2016.

[153] Tianyi Zhou and Jeff Bilmes. Minimax curriculum learning: Machine teaching with desirable difficulties and scheduled diversity. In *ICLR*, 2018.

[154] Tianyi Zhou, Shengjie Wang, and Jeff A Bilmes. Diverse ensemble evolution: Curriculum data-model marriage. In *Advances in Neural Information Processing Systems 31*, pages 5905–5916. 2018.

[155] Zeyuan Allen Zhu, Yang Yuan, and Karthik Sridharan. Exploiting the structure: Stochastic gradient methods using raw clusters. In *NIPS*, 2016.

[156] Jingwei Zhuo, Jun Zhu, and Bo Zhang. Adaptive dropout rates for learning with corrupted features. In *IJCAI*, pages 4126–4132, 2015.

[157] Konrad Zolna, Devansh Arpit, Dendi Suhubdy, and Yoshua Bengio. Fraternal dropout. In *ICLR*, 2018.