Learning Robust Tractable Models for Vision

Robert Gens

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

Doctor of Philosophy

University of Washington

2016

Reading Committee:

Pedro Domingos, Chair

Ali Farhadi

John Platt

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Learning Robust Tractable Models for Vision

Robert Gens

Chair of the Supervisory Committee:
Professor Pedro Domingos
Computer Science and Engineering

Human vision is a demanding computation that acts on and learns from billions of moving measurements every second. Computer vision requires models that are both tractable for realtime learning and inference as well as robust to the transformations of the visual world.

For a vision system to benefit an embodied agent it must be able to (a) learn tractable models discriminatively so that it does not waste computation on nonessential questions, (b) learn model structure so computation is only added where needed, (c) learn from images subject to transformations, and (d) learn new concepts quickly. In this dissertation we tackle these four desiderata, weaving together sum-product networks, neural networks, kernel machines, and symmetry group theory.

First, we extend sum-product networks so that they can be trained discriminatively. This expands the space of SPN architectures and allows feature functions, making them a compelling tractable alternative to conditional random fields. We show that discriminative SPNs can be competitive with deep models on image classification.

Second, we present an algorithm to learn the structure of sum-product networks. The top-down recursive algorithm builds a product if it can decompose variables and otherwise a sum to cluster instances. Surprisingly, this algorithm learns SPNs with superior inference accuracy and speed compared to probabilistic graphical models on a large number of datasets.

Third, we introduce deep symmetry networks that can learn representations over arbitrary

Lie groups. We present techniques to scale these networks to high-dimensional symmetries. We show that deep symmetry networks can classify 2D and 3D transformed objects with higher accuracy and less training data than convolutional neural networks.

Finally, we propose compositional kernel machines as an instance-based learner that has the symmetry and compositionality of convolutional neural networks but is significantly easier to train. We combat the curse of dimensionality by effectively summing over an exponential set of constructed virtual training instances using a sum-product function. This makes CKMs outperform standard instance-based learners on image classification and generalizing to unseen compositions and symmetries.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am tremendously grateful to everyone who brought me here.

First, I would like to thank my advisor, Pedro Domingos. From day one, he took me and my strange interest in neuroscience seriously and taught me how to translate audacious goals into a research agenda. When I play Hearts with my family, I stubbornly attempt to "shoot the moon" every hand. Pedro does the same with research, only aiming at the hardest problems, which is why we get along. I am most fortunate to have been guided by his broad perspective, humor, and optimism.

I am grateful to have the guidance of a committee that fits this dissertation like a glove. For a number of years, Richard Newcombe was practically a second advisor, spending inordinate amounts of time zealously discussing the latest research. Without him, the Deep Symmetry Networks would be literally stuck in local minima. During my summer internship at MSR, I greatly benefitted from the friendship and perspective of John Platt. His dual-citizenship with connectionism and kernel methods was invaluable in forming Compositional Kernel Machines. I look to Ali Farhadi's research for a far-sighted view of structured vision problems. His support and keen feedback have been essential to this work. I was inspired by Marina Meila's early work on tractable probabilistic models and am grateful that her clear explanations of kernel methods and optimization have stuck with me.

I would like to thank my mentors outside of UW. I would like to extend special thanks to Robert Peharz with whom I collaborated and had many great discussions. I am honored to have the support and superb feedback of Yoshua Bengio, whose research and magnanimity have fostered the research community in which I work. I am also privileged to have discussed my research with Geoffrey Hinton on several occasions. I wish to thank Patrice Simard for

sharing his research insights with me.

I was lucky to know the students and faculty at UW CSE. My groupmates Vibhav Gogate, Chloé Kiddon, Stanley Kok, Daniel Lowd, Xu Miao, Aniruddh Nath, Mathias Niepert, and Hoifung Poon have been tremendously helpful and generous with their time. I especially want to thank Jesse Davis for teaching me how to run thousands of experiments and for showing up years later at the Snowbird Workshop to motivate me to continue the discriminative learning project. I am profoundly thankful to my labmates Abram Friesen and Rahul Kidambi, whose quick technical understanding, patience, and mental clarity have impacted my work. I am grateful to my officemates Bilge Soran, Ezgi Mercan, and Michael Chung for their friendship, energy, and feedback. I wish to thank Mark Yatskar for many helpful discussions and hoisting my experiments into the cloud. I am indebted to Kendall Lowery for his infinite kindness and creative research proposals. I was fortunate to make great friends with talented people: Nicholas FitzGerald, Ricardo Martin, Tony Fader, Adrienne Wang, Eunsol Choi, Tom Kwiatkowski, Neeraj Kumar, Tom Erez, Yuval Tassa. I have learned from each of them. Yoav Artzi and Julija Lazutkaite made Seattle a special place full of laughter, food, nature, and aesthetically pleasing photographs. UW CSE is truly a one-of-a-kind department. I would like to express my gratitude to Luke Zettlemoyer, Dieter Fox, Dan Weld, Carlos Guestrin, Emily Fox, and Hank Levy for their support and candor. I thank Lindsay Michimoto, Elise DeGoede, Patrick Allen, and Andrei Stabrovski for making my graduate experience as painless as possible. UW boasts an impressive computational neuroscience community. I wish to thank Rajesh Rao, Adrienne Fairhall, and Eric Shea-Brown for engaging me in great discussions.

I would not be here without teachers who believed in me. I am forever grateful to Ted Selker, who launched me into the research world with the opportunity to work in his MIT lab as a high school student; I continue to be inspired by his creativity and perspicacity. I am thankful to Frédo Durand and Ramesh Raskar who provided independent research

opportunities during my undergrad studies. Ben McGraw, Theodore May, Murph Shapiro, Neel Pandeya, Rosa Sterk, and Linda Jordan Kraus receive my deepest gratitude for taking a special interest in my education.

My odyssey would not have been possible without my family. My parents Dianne and Barry gave me the freedom, resources, confidence, and curiosity to pursue whatever I wanted. I am supremely grateful that they encouraged me to break, fix, glue, build, solder, and program things at such a young age. I am lucky to have my brothers Matthew and Jason for support, critique, and unceasing iterations of jokes. To my wife, Melanie, thank you. I am humbled by your intelligence and character. Thank you for the adventures both at home and far-flung islands "where there [are] many sea wolves and large birds". Thank you for being my non-drowsy habit-forming pick-me-up on this challenging journey. I would not have been able to do this without you.

My graduate studies and projects were supported in part by a Google PhD Fellowship, an Amazon AWS in Education Grant, an NVIDIA academic hardware grant, the Corin Anderson CSE Endowed Fellowship, the UW College of Engineering Egtvedt Fellowship, and funding from the ARO and ONR.

DEDICATION

To my grandparents Burt, Esther, Julius and Marilyn, who inspire me to build a better world.

Chapter 1

INTRODUCTION

Images present the ultimate challenge for machine learning: searching for meaning and making predictions from millions of noisy variables. The megapixels in an image are not truly random; they are the result of a much smaller number of variables that influence the physical process of image formation. The goal of representation learning is to resolve these semantic variables automatically from data without feature engineering. In a deep learning system the representation is staged in layers of non-linear functions or latent variables in probabilistic models. A deep model can more compactly encode a target function than a network with fewer layers. It also maps to a hierarchical part-based modeling of the visual world. The two major categories of deep architectures are generative and discriminative.

Generative models are desirable because they learn how all variables relate to each other, can handle missing data such as with occlusion, and do not require labeled data. Examples include deep Boltzmann machines, deep belief networks, and certain types of autoencoders. These approaches typically involve probabilistic graphical models and thus inherit intractable inference, meaning that the time it takes to answer a question can grow exponentially with the number of variables. This is problematic for deep models where upwards of thousands of latent variables are used to model data. To cope, one must either restrict the model to inexpressive tree structures or use approximate inference, which is unreliable and can interfere with learning. Recently Poon and Domingos introduced sum-product networks, a class of models where adding layers increases expressiveness without losing tractability. Sum-product networks achieved impressive results on a challenging image completion task. However, their structure had to be designed by hand. This dissertation fills this gap by contributing a general algorithm to learn the structure of SPNs.

Discriminative models are state of the art for many domains because they are optimized for specific tasks. Rather than modeling all dependencies among variables, they adjust their parameters to improve the prediction of a subset of the variables for classification, regression, or structured prediction. Conditional random fields are discriminatively trained probabilistic graphical models commonly used in vision to predict structured outputs, however they also suffer from intractable inference. Sum-product networks are well-suited for this task, but they could only be trained generatively. This dissertation addresses this problem by presenting a discriminative training algorithm for SPNs. Discriminative instance-based learning (IBL) methods such as k-nearest neighbors and support vector machines are easier to train than neural networks; however, they presently are not as accurate as these deep methods. Discriminative deep models include multilayer perceptrons, convolutional neural networks, recurrent neural networks, and long short term memory networks. A limited ability to generalize to complex transformations and costly non-convex optimization are some of the weaknesses of these methods. This dissertation proposes deep symmetry networks and compositional kernel machines to mitigate these issues.

In this dissertation, we present the first discriminative training algorithms for SPNs, combining the high accuracy of the former with the representational power and tractability of the latter. We show that the class of tractable discriminative SPNs is broader than the class of tractable generative ones, and propose an efficient backpropagation-style algorithm for computing the gradient of the conditional log likelihood. Experiments on standard image classification tasks show that discriminatively trained SPNs are competitive with state of the art neural networks.

Though SPNs have demonstrated success learning weights on pre-defined structures, learning deep structure from data is important for several reasons. Firstly, static architectures waste computation on nodes they do not need. Secondly, hand-designed architectures might not contain the best nodes or connections for a given dataset. Finally, in the case of SPNs, tractability and expressivity comes with more intricate network architecture which may be more difficult or time-consuming to design by hand. We propose the first algorithm

for learning the structure of SPNs that takes full advantage of their expressiveness. At each step, the algorithm attempts to divide the current variables into approximately independent subsets. If successful, it returns the product of recursive calls on the subsets; otherwise it returns the sum of recursive calls on subsets of similar instances from the current training set. A comprehensive empirical study shows that the learned SPNs are typically comparable to graphical models in likelihood but superior in inference speed and accuracy.

Learning a probabilistic model over data is easier if its variables stay put from instance to instance. However, no two images are the same due to a large number of extraneous sources of variability, such as pose and part deformation. These sources of variation can be represented by symmetry groups, sets of composable transformations that preserve object identity. An important goal for representation learning is to be able to learn features that retain their semantic content while being invariant to these transformations. Convolutional neural networks (convnets) achieve a degree of translational invariance by computing feature maps over the translation group, but cannot handle other groups. As a result, these groups' effects have to be approximated by small translations, which often requires augmenting datasets and leads to high sample complexity. In this dissertation, we present deep symmetry networks (symnets), a generalization of convnets that forms feature maps over arbitrary Lie groups. The composition of feature transformations through the layers of a symnet provides a new approach to deep learning. Experiments on NORB and MNIST-rot show that symnets over the affine group greatly reduce sample complexity relative to convnets by better capturing the symmetries in the data.

For a visual representation to be useful in embodied or interactive settings it must be able to learn from new examples quickly. Even with static datasets, training convnets is time-consuming because stochastic gradient descent requires many passes over the training data which is already large because of data augmentation. This procedure is also frustrating because several modifications are required to achieve top results, including momentum, learning rate schedules, regularization, dropout, and more recently batch normalization. Compounding the overall time and computation, the search over hyperparameters is a massive outer

loop around parameter fitting. In contrast, instance-based learners (IBL) have much simpler training procedures (or none at all) and can more efficiently search over hyperparameters. However, in high dimensions (i.e. images) the concept of a nearest neighbor breaks down as most instances are equally distant. An IBL that incorporates the composition and symmetry of convnets could mitigate this issue. We present compositional kernel machines (CKMs), which effectively create an exponential number of virtual training instances by composing transformed sub-regions of the original ones. Despite this, CKM discriminant functions can be computed efficiently using ideas from sum-product networks. The ability to compose virtual instances in this way gives CKMs invariance to translations and other symmetries, and combats the curse of dimensionality. In this dissertation we present efficient techniques to add image composition, symmetry, and weight learning to CKMs. Experiments show that CKMs are superior to standard instance-based methods and can learn symmetries and compositional concepts from fewer samples without data augmentation.

The structure of this dissertation is as follows. Chapter 2 covers background material on probabilistic graphical models, deep learning, instance-based learning, and sum-product networks. Chapter 3 presents an algorithm to discriminatively train sum-product networks. Chapter 4 proposes an algorithm to learn the structure of sum-product networks from data. Chapter 5 presents Deep Symmetry Networks as a generalization of convolutional neural networks to arbitrary Lie groups. Chapter 6 introduces Compositional Kernel Machines as a promising visual representation with the benefits of instance-based learning. The last chapter summarizes our contributions and suggests promising directions for future research.

Chapter 2

BACKGROUND

2.1 Probabilistic Graphical Models

Probabilistic graphical models provide a compact graph-based factorization of the probability distribution for a set of variables. In the simplest case, nodes in the graph correspond to variables and edges indicate a dependency between variables. The two main types of graphical models are Bayesian networks with directed acyclic graphs and Markov networks with undirected graphs [112]. In a graphical model over a set of variables $\mathbf{X} = (X_1, \ldots, X_n)$, the joint probability of a setting of those variables $\mathbf{X} = \mathbf{x}$ is written as

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_i \phi_i(\mathbf{x})$$

where the potential functions $\phi_i(\mathbf{x})$ are determined by the graph structure and whether the model is Bayesian (one potential for the conditional probability of each variable given its parents in the directed graph) or Markovian (one potential for each maximal clique in the undirected graph). The partition function Z is the sum of probabilities over all possible variable states $Z = \sum_{\mathbf{x}} \prod_i \phi_i(\mathbf{x})$, which conveniently equals one for Bayesian networks because the potentials are conditional probabilities.

Probabilistic inference computes the conditional probability of a query given evidence in the form $P(\mathbf{Q} = \mathbf{q} | \mathbf{E} = \mathbf{e})$, where query variables $\mathbf{Q}$ and evidence variables $\mathbf{E}$ are disjoint subsets of the model variables $\mathbf{X}$. The exact computation of this conditional probability requires summations over the states of other variables

$$P(\mathbf{Q} = \mathbf{q} | \mathbf{E} = \mathbf{e}) = \frac{\sum_{\mathbf{X} \setminus (\mathbf{Q} \cup \mathbf{E}) = \mathbf{u}} \prod_i \phi_i(\mathbf{q}, \mathbf{e}, \mathbf{u})}{\sum_{\mathbf{X} \setminus \mathbf{E} = \mathbf{v}} \prod_i \phi_i(\mathbf{e}, \mathbf{v})}$$

which is #P-complete [127].

Exact inference is tractable for graphical models with low treewidth [7, 24]. This condition essentially limits dependency structures to chains and trees, which is inexpressive for most applications and antithetical to the designs of most deep probabilistic models [9].

Approximate inference in graphical models is most often achieved with sampling or message-passing techniques [80]. With Bayesian networks, one can directly generate samples from a conditional distribution that may be used for Monte Carlo (MC) inference; however, this procedure can be quite costly when there are many evidence variables. For both Markov and Bayesian networks, Markov chain Monte Carlo (MCMC) is used to produce a sequence of samples that ideally approaches the desired distribution. The most common MCMC method is Gibbs sampling, which generates a subsequent sample by iterating over all variables, sampling each conditioned on its Markov blanket. MCMC is more computationally-demanding than MC because to prevent correlation only a small fraction of the generated samples is retained. It is also difficult to diagnose convergence of the Markov chain.

Belief propagation is also used for approximate inference in graphical models. There are several variants where message passing is conducted on either the original graph, a factor graph, or random trees that span the graph [107, 161]. Belief propagation, however, only estimates single variable marginals and there are no guarantees of convergence. Interestingly, for stereo vision problems these techniques have been known to achieve energies that are lower than those of the ground-truth depth labeling, suggesting that there may be other probabilistic topologies better suited for images [151].

With complete data, learning the maximum-likelihood parameters of a graphical model can be achieved in closed form for Bayesian networks or with gradient-based optimization for Markov networks. With incomplete data or latent variables, the expectation-maximization algorithm or gradient methods are used. For all parameter learning scenarios except for Bayesian networks with complete data, inference is a costly subroutine, and approximate inference can interfere with learning [85]. The structure of graphical models can also be learned from data. Most algorithms involve a greedy search that evaluates the impact of

modifying edges in the graph on likelihood or a proxy [43, 26]. To prevent overfitting, likelihood is usually augmented by other criteria such as the number of parameters or inference complexity [98].

2.2 Deep Learning

Deep Learning is a collection of techniques for learning multiple levels of a representation with non-linear functions or latent variables [9]. The two main categories of deep networks are discriminative and generative models. Discriminative methods compare the output of an artificial neural network with the desired outcome to a classification or regression task and adjust weights to reduce error. Generative methods do not require labeled data and attempt to automatically learn the latent representation of data.

2.2.1 Generative Models

For connectionists who are inspired by biology, unsupervised learning is appealing because the world is not neatly organized as (x, y) pairs of data and labels. The Boltzmann machine was an early generative connectionist model proposed by Ackley et al. [3]. It is a fully-connected Markov random field with visible $\mathbf{V}$ and hidden $\mathbf{H}$ variables. The form of the potential that connects each pair of variables X_i, X_j in a binary Boltzmann machine is $\phi_{ij}(x_i, x_j) = e^{w_{ij}x_i x_j}$, where $x_i \in \{0, 1\}$. Intuitively, a large weight leads to a higher unnormalized probability if both variables are true. Each variable also has a singleton potential $\phi_i(x_i) = e^{w_i x_i}$ as a bias or prior. The weights are optimized to maximize the probability of the observed data. The gradient of a weight is the difference between data and model expectations of the state of the connected variables: $\frac{\partial P(\mathbf{V})}{\partial w_{ij}} = \langle x_i x_j \rangle_{data} - \langle x_i x_j \rangle_{model}$. Since there are hidden variables, both terms require inference, which must be approximated. Gibbs sampling is typically used, but it is very slow.

Restricted Boltzmann machines (RBMs) [143] take the form of a bipartite graph where all visible variables connect with all hidden variables but no variable connects to a variable of its own type. This allows for faster blocked Gibbs sampling, where conditioned on visible

units, hidden variables can be sampled simultaneously and vice-versa. Another simplification called "contrastive divergence" involves using an early sample from the Markov chain instead of waiting for convergence [67].

Deep belief networks (DBNs) [68] and deep Boltzmann machines (DBMs) [131] are multi-layer generative models formed by stacking RBMs. The former is a directed model and the later is an undirected Markov network. Both are greedily pretrained layer-by-layer. After pretraining an RBM layer, the weights are fixed and a new representation of the data is created from samples or expectations of the hidden units conditioned on the visible units. This is then used to pretrain the next layer. Since exact inference is intractable, a combination of variational mean field and MCMC inference is used for the data and model terms of the gradient, respectively.

2.2.2 Discriminative Models

One of the first artificial neuron models proposed by McCulloch and Pitts [103] involved the weighted combination of binary inputs that was compared against a threshold to fire a binary output. The perceptron learning algorithm was proposed by Rosenblatt [126], but it could only train a network with a single layer. The basic form of the artificial neuron still used today is

$$o = \sigma(\vec{w} \cdot \vec{x})$$

where $\vec{x}$ and $\vec{w}$ are input and weight vectors, and σ is known as the transfer or activation function. Historically, the activation function has been a sigmoid ($\sigma(x) = 1/(1 + e^{-x})$) or hyperbolic tangent function.

Multi-layer perceptrons (MLPs) are trained using the backpropagation learning algorithm, which computes the gradient of lower-level parameters using the chain-rule of calculus [163, 130]. Training networks with more than a few layers can be problematic because of the "vanishing gradient" problem where the learning signal would shrink and become diffuse as it is computed top down, at each layer multiplied by the fractional weights and the fractional

slope of a sigmoid [70, 11]. Though layer-by-layer unsupervised pre-training was explored as a possible remedy [68], state-of-the-art neural networks for object recognition still use random weight initialization and backpropagation. The issue of vanishing gradients has been mitigated by rectified linear unit transfer functions, weight-tying, and faster computers [134]. At the price of longer training times, the dropout technique stochastically disables a fraction of the neurons in order to help prevent overfitting [84].

Neuroscience gave artificial neural networks one of its most effective tools for addressing symmetries. Hubel and Wiesel [72] found a hierarchy of "simple" and "complex" neurons in the visual cortex of the cat. Whereas the simple cells were activated by edges at specific positions and orientations, the complex cells would fire preferentially for edges of a certain orientation regardless of its position within a range of the visual field. This inspired Fukushima [56] to devise a neural network architecture with complex cells that would sum over the outputs of simple cells that measured the response to the same pattern in several locations. LeCun et al. [90] would later create the convolutional neural network that shared the pooling property of the complex cells but used backpropagation for learning to classify handwritten digits. The convnet LeNet-1 performed well on the MNIST dataset but was soon surpassed by nearest neighbor using tangent distance [138] and SVMs with virtual support vectors [135]. Later, the addition of more convolutional layers and substantial data augmentation would allow convnets to tie and then exceed the accuracy of SVMs [91, 139].

The computer vision community largely ignored convnets until Krizhevsky et al. [84] demonstrated state-of-the-art results on the ImageNet LSVRC-2012 challenge. The main innovations of this work were the use of rectified linear units, dropout regularization, and the use of graphics processing units (GPUs). Since then, the community has been continually finding architectures with better performance. Salient trends include the use of extra layers of 1×1 convolutional kernels [140] to increase non-linearity and architectures with trained outputs at several layers [148]. Open source software with pre-trained models has led to widespread use of convnets (e.g., [77, 33, 1]). However, training a model from scratch can still take several days, even on large clusters.

2.3 Instance-Based Learning

Instance-based learning (also known as lazy learning) is a family of methods that stores whole training instances and does not explicitly train a model. At test time, a subset of training instances that are similar to the test instance are recruited to form a local decision (e.g., majority vote classification for k-nearest neighbors). Despite their simplicity, IBL methods can implicitly represent very sophisticated non-linear concepts, and historically they have been competitive with neural networks. Two main categories of IBL techniques used for vision tasks are nearest neighbor and support vector machines.

2.3.1 Nearest Neighbor

The nearest neighbor (NN) classifier labels a test instance with the class of the training instance whose features are closest using a specified distance function [35]. The k-NN variant uses the majority vote of the k nearest neighbors. The greatest advantage of k-NN is that training is very simple and yet can construct very sophisticated nonlinear decision boundaries local to each query point. In the limit of infinite data it has an error that is no worse than the Bayes error rate [35]. A drawback is that it requires the storage and search computation for the entire training set to classify a test instance. It is also susceptible to being swayed by uninformative features, which only gets worse in high dimensions.

Various techniques have been developed to cope with irrelevant features. Feature relevance can be measured in isolation using information gain. Since leave-one-out cross-validation is efficient with k-NN, it can be used to decide which dimensions to omit [94] or to scale [105]. However, these methods involve a single diagonal linear projection, which does not leverage the correlation among pixels within natural images.

The most common method of dimensionality reduction for nearest neighbor is principal component analysis, which computes a linear projection that preserves the variance of the data. Sirovich and Kirby [141] applied this to 128×128 images of faces, finding that most of the variance could be explained by fewer than a hundred components. Turk et al. [157]

extended this work to face recognition by using a nearest neighbor classifier on this reduced "facespace". The linear understanding of the face was easily confused by background clutter, where an image patch that had darker pixels for "eyes" and "mouth" could additively compensate for the complete lack of a nose. Sung and Poggio [146] tried to capture this non-linear boundary by fitting two mixtures of six Gaussians to a large set of face and non-face patches, where the distance of a test image to a Gaussian cluster was measured in the low-dimensional space spanned by the cluster's top eigenvectors. The idea that distances should be measured to the closest point on a low-dimensional non-linear manifold of transformed images was crystallized by Bregler and Omohundro [20]. Their technique involves computing PCA on the nearest neighbors of a data point to estimate the local dimensionality of the manifold. This approach is useful for deciding which of several labeled manifolds is closest to a test point, but it does not have the ability to unite the many locally-linear systems into low-dimensional global coordinates as later accomplished by Tenenbaum et al., Roweis and Saul [152, 128]. Both sets of techniques are sample-intensive and cannot generalize to image symmetries.

Computing distances using a local subset of pixels greatly reduces the dimensionality of images. Of course, if training data is annotated to reveal relevant dimensions, the distance computation is more discriminative. Pentland et al. [115] found that an eigenspace analysis of images cropped to a particular face region (e.g., eyes) yielded robust detections of those parts in test images. Features such as textons [79], visual words (e.g., [142]), SIFT [99], and HOG [37] are all extracted locally and have been used successfully with nearest neighbor and other classifiers. It has been argued that nearest neighbor would be an ideal classifier for these features if it were not for feature quantization and computation of image-to-image distance [17]. Local naive Bayes and its extensions [102, 154] share the non-parametric motivations of this dissertation but do not propose any composition of features, learning algorithms, or techniques to model non-trivial symmetries.

A key problem with object recognition is that these local regions or parts can vary in their location or appearance due to nuisance transformations. Pictorial structures [52] was one

of the first principled approaches to computing a distance that incorporated both the similarity of local regions to templates as well as penalized elastic distances between templates. Simplifying the structure of this model to arrange part-to-part relations in a tree made for a tractable dynamic program and allowed the use of efficient distance transform computations [51]. When computing the distance to the image manifold due to other symmetries, a dense sampling is usually avoided. Simard et al. [138] approximate this distance using the distance to a hyperplane tangent to the manifold of several image distortions. Iterative optimization in symmetry space is commonly used to compute similarity [8]. Using properties of symmetry groups leads to an alignment algorithm that avoids the local minima of Lucas-Kanade optimization yet does not require a dense sampling of symmetry space [153].

2.3.2 Kernel Methods

Sparse kernel machines address many of the weaknesses of nearest neighbor and neural network classifiers. This family of approaches arose from research on statistical learning theory by Vapnik [159] and Vapnik and Kotz [160], asking how the complexity of a learner impacts its ability to generalize to unseen data. The problem with the perceptron [126] and multi-layered perceptrons [130] was that there was no guarantee that minimizing training error would minimize test set error. Furthermore, it is intractable to use these methods on data that has been mapped to a high dimensional feature space (e.g., polynomials). Boser et al. [19] proposed a technique that could find max-margin hyperplanes with favorable generalization properties. It leveraged the kernel trick that had allowed Poggio [119] to efficiently perform least squares regression with polynomial features. The inability to handle non-separable data was addressed by the support-vector network [34]. From the perspective of its dual form, the SVM can be seen as a weighted nearest neighbor with superior generalization, ability to tolerate noise, and economical yet discriminative storage of exemplars. These abilities would otherwise require heuristics for nearest neighbor (e.g., [63, 164]). Other sparse kernelized classifiers have been formulated since (e.g., [155, 83]), but SVMs remain the most popular. Since the original quadratic program was proposed, more efficient learning

algorithms have been researched (e.g., [117, 78, 18, 71, 137]) with qualities that can rival or surpass those of the stochastic gradient descent used by neural networks.

Some of the earliest research into SVMs involved making them robust to image transformations [135, 42], which introduced techniques that made SVMs competitive with neural networks on MNIST. Rather than augmenting the training set with a set of transformations, they proposed expanding the set of support vectors into a set known as *virtual support vectors*. This reduces training time but increases test time evaluation. Rather than explicitly create the set of virtual support vectors, Decoste and Schölkopf [42] describe a jittered kernel that searches for a transformation that minimizes the distance between two examples. Unlike the CKMs we propose, jittered kernels carry the limitations of image-to-image distances and they do not propose an efficient method for searching over transformations. Whereas the relative transformations between training set images with a jittered kernel are fixed, the latent SVM deformable parts model (DPM) [50] introduces latent variables over part positions and a fast method for aligning these parts with learned location constraints. Compared to CKMs, DPMs have a fairly restrictive sense of image composition and suffer from a long CCCP (EM-style) training procedure to find the latent positions of a fixed number of parts which might not exist.

Various kernels have been used to compare images. Early work used polynomial kernels that leveraged image locality [136]. However the advent of sparse invariant image features (e.g., Harris, Jet [133], SIFT [99]) changed the research from measuring similarity of dense transformed pixel grids or intensity histograms to that of variable-sized sets of features. For a pair of images, match kernels [162] compute the sum over the similarities of each feature in one image and its closest match in the other: $\hat{K}(L_h, L_k) = \frac{1}{n_h}\Sigma_{j_h=1}^{n_h}\max_{j_k=1,\ldots n_k}\{K_l(l_{j_h}(L_h), l_{j_k}(L_k))\}$, where n_h, n_k are the numbers of features in images L_h, L_k respectively and $K_l(.,.)$ is a Mercer kernel over features. Since these correspondences are computed independently, image composition is not preserved or accounted for in the kernel. The pyramid match kernel [62] seeks to alleviate this weakness by aggregating features according to several quantizations and measuring the histogram intersection of correspond-

ing grid cells. This kernel performs better than the match kernel because it measures feature co-occurrence, but binning by quantization does not preserve spatial relations among features. Rather than build pyramids that subdivide each element of a d-dimensional feature, spatial pyramids [89] construct pyramids in the traditional sense (e.g., [22]) by subdividing the two-dimensional image plane into histogram bins that count the occurrence of M discrete visual words. At the time, this method achieved nearly state of the art results on the Caltech-101 dataset. However, it was argued that feature quantization by hard assignment was hurting performance [17].

Fisher kernels [75] adapt generative models into a kernel using the gradient of the model's log-likelihood with respect to its parameters. When used with Gaussian mixture models, the Fisher kernel provides a soft encoding and incorporates a learned covariance compared to the hard assignment and spherical Euclidean distance typical of visual words. Combined with spatial pyramids, Fisher kernels had formed the basis for state-of-the-art image classification techniques [116]. Spatial pyramids pool locally, so a coherent assembly of features that translates a large distance from training to test images will not have the benefit of being matched in finer pyramid cells. Convolutional neural networks also pool locally, but the convolved features are learned from all parts of the feature map, so the matching is not restrained as in a spatial pyramid. With this difference in pooling and the benefit of discriminatively-trained features, convnets overtook SVMs for object recognition in the work of Krizhevsky et al. [84].

2.4 Sum-Product Networks

Sum-product networks are a new type of probabilistic deep network with tractable inference on high-treewidth models [120]. In the simplest instance, it is a directed acyclic graph with sum and product internal nodes and tractable distributions[1] at the leaves. Sum nodes are viewed as the marginalization of hidden mixture variables. Seen bottom-up, product nodes are conjunctive features; top-down, they represent context-specific independences. There are

[1] A distribution is tractable iff its partition function and its mode can be computed in constant time.

two ways to define SPNs: one more general but complex [120], and one more specialized but simple [58]. We describe the latter here for clarity and present the former in Chapter 3 where it is useful for derivation and discriminatively-trained architectures.

Generative probabilistic SPNs can be defined recursively: (1) a univariate distribution is a SPN, (2) the product of SPNs over disjoint sets of variables is an SPN, (3) the weighted sum of SPNs over the same set of variables is an SPN. Discriminative probabilistic SPNs that compute $P(Y|X) = \Sigma_h P(Y, H = h|X)$ do not need to obey the constraints over the scopes of given variables X [57].

The sub-SPN S_i rooted at a node i represents a probability distribution over its scope. For simplicity, we focus on the case of SPNs over discrete variables, but the extension to continuous ones is straightforward.

Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathcal{X}$ be a state. The unnormalized probability $S(\mathbf{x})$ of $\mathbf{x}$ according to the SPN S is the value of S's root when each leaf is set to the probability of the corresponding variable's value in $\mathbf{x}$. The partition function of an SPN is $Z = \sum_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x})$. The normalized probability of $\mathbf{x}$ is $P(\mathbf{x}) = S(\mathbf{x})/Z$. It is easily seen that, if the weights at each sum node sum to one and the leaf distributions are normalized, then $Z = 1$ and $P(\mathbf{x}) = S(\mathbf{x})$.

Theorem 1. *The following quantities can be computed in time linear in the number of edges in an SPN.*

1. *The partition function of the SPN.*
2. *The probability of evidence in the SPN.*
3. *The MAP state of the SPN.*

Proof. The proof is recursive, starting from the leaves of the SPN. By definition, the partition function of a leaf distribution can be computed in $O(1)$ time. Let Z_i be the partition function of node i, and Z_{ij} the partition functions of its children. Let $\mathcal{X}_i$ be the set of possible states of i's scope, and similarly for $\mathcal{X}_{ij}$. If i is a product node, its partition function is $Z_i = \sum_{\mathbf{x}_i\in\mathcal{X}_i} S_i(\mathbf{x}_i) = \sum_{\mathbf{x}_{i,1}\in\mathcal{X}_{i,1}} \cdots \sum_{\mathbf{x}_{ij}\in\mathcal{X}_{ij}} \cdots \; S_{i,1}(\mathbf{x}_{i,1}) \ldots S_{ij}(\mathbf{x}_{ij}) \ldots = \prod_j \sum_{\mathbf{x}_{ij}\in\mathcal{X}_{ij}} S_{ij}(\mathbf{x}_{ij}) = \prod_j Z_{ij}$. If i is a sum node, its partition function is $Z_i = \sum_{\mathbf{x}_i\in\mathcal{X}_i} S_i(\mathbf{x}_i) = \sum_{\mathbf{x}_i\in\mathcal{X}_i} \sum_j w_{ij} S_{ij}(\mathbf{x}_i) =$

$\sum_j \sum_{\mathbf{x}_i \in \mathcal{X}_i} w_{ij} S_{ij}(\mathbf{x}_i) = \sum_j w_{ij} \sum_{\mathbf{x}_{ij} \in \mathcal{X}_{ij}} S_{ij}(\mathbf{x}_{ij}) = \sum_j w_{ij} Z_{ij}$, where w_{ij} is the weight of the ith child. Therefore the partition function of a node can be computed in time linear in its number of children, and the partition function of the root can be computed in time linear in the number of edges in the SPN.

The probability of evidence in an SPN S is just the ratio of the partition functions of S' and S, where S' is an SPN obtained from S by replacing the univariate distributions over the evidence variables by delta functions centered on the evidence values. Therefore it can also be computed in linear time.

The (or a) MAP state of an SPN can be computed as follows: (1) replace sum nodes with max nodes; (2) evaluate the SPN from the leaves to the root in a manner identical to computing the partition function; (3) starting from the root and following all children of each product node, for each sum node S_i choose the (or a) child with highest value of $w_{ij} M_{ij}$, where M_{ij} is the child's value computed in the previous step; (4) at each leaf node, choose the (or a) mode of the corresponding distribution. The total number of operations is thus also linear in the size of the SPN. □

SPNs have many interesting and important classes of probabilistic models as special cases, including mixture models, thin junction trees, non-recursive probabilistic context-free grammars, and others. They are also significantly more general than each of these. SPNs' combination of expressiveness and tractability makes them potentially a very attractive representation for many applications. The same algebraic decomposition that yields tractable inference for SPNs on the sum-product semiring applies to *sum-product functions* over other semirings impacting many other tasks including integration, optimization, and constraint satisfaction [54]. For example, it has been applied to non-convex optimization on the min-sum semiring with impressive results on bundle adjustment and protein folding [53]. We use sum-product networks as tractable probabilistic models in Chapters 3 and 4 and then sum-product functions as tractable kernel machines in Chapter 6.

Chapter 3

DISCRIMINATIVE LEARNING OF SUM-PRODUCT NETWORKS

3.1 Introduction

Poon and Domingos introduced an algorithm for generatively training SPNs, yet it is generally observed that discriminative training fares better for structured prediction. By optimizing $P(\mathbf{Y}|\mathbf{X})$ instead of $P(\mathbf{X}, \mathbf{Y})$ conditional random fields retain joint inference over dependent label variables $\mathbf{Y}$ while allowing for flexible features over given inputs $\mathbf{X}$ [87]. Unfortunately, the conditional partition function $Z(\mathbf{X})$ is just as prone to intractability as with generative training. For this reason, low treewidth models (e.g. chains and trees) of $\mathbf{Y}$ are commonly used. Multilayered perceptrons and energy-based models [93] can also be used to produce structured predictions, but they cannot guarantee exact inference. Research suggests that approximate inference can make it harder to learn rich structured models [85]. In this chapter, we present the first discriminative training algorithms for SPNs to allow us to combine flexible features with fast, exact inference over high treewidth models. We begin with a review of the original definition of SPNs using indicator notation. We then describe the conditions under which an SPN can represent the conditional partition function. We show that the class of tractable discriminative SPNs is broader than the class of tractable generative ones because evidence variables are not marginalized. We then propose an efficient backpropagation-style algorithm for computing the gradient of the conditional log likelihood, and explore variations of inference. Finally, we show results where a discriminatively-trained SPN achieves higher accuracy than SVMs and deep models on image classification tasks. At the time of publication, the results on CIFAR-10 and STL-10 were state-of-the-art, which was notable because the SPN was using fewer features than prior methods and much less

training data in the case of STL-10.

3.2 Sum-Product Networks

SPNs were introduced with the aim of identifying the most expressive tractable representation possible. The foundation for their work lies in Darwiche's network polynomial [38]. Instead of the definition of SPNs presented in the previous chapter, we use the original definition here as its precision is useful for mathematical derivation. We define an unnormalized probability distribution $\Phi(\mathbf{x}) \geq 0$ over a vector of Boolean variables $\mathbf{X}$. The indicator function [.] is one when its argument is true and zero otherwise; we abbreviate $[\mathrm{X_i}]$ and $[\bar{\mathrm{X}}_\mathrm{i}]$ as x_i and $\bar{x}_i$. To distinguish random variables from indicator variables, we use roman font for the former and italic for the latter. Vectors of variables are denoted by bold roman and bold italic font, respectively. The network polynomial of $\Phi(\mathbf{x})$ is defined as $\sum_{\mathbf{x}} \Phi(\mathbf{x}) \prod(\mathbf{x})$, where $\prod(\mathbf{x})$ is the product of indicators that are one in state $\mathbf{x}$. For example, the network polynomial of the Bayesian network $\mathrm{X_1} \rightarrow \mathrm{X_2}$ is $P(\mathrm{x_1})P(\mathrm{x_2}|\mathrm{x_1})x_1x_2 + P(\mathrm{x_1})P(\bar{\mathrm{x}}_2|\mathrm{x_1})x_1\bar{x}_2 + P(\bar{\mathrm{x}}_1)P(\mathrm{x_2}|\bar{\mathrm{x}}_1)\bar{x}_1x_2 + P(\bar{\mathrm{x}}_1)P(\bar{\mathrm{x}}_2|\bar{\mathrm{x}}_1)\bar{x}_1\bar{x}_2$. To compute $P(\mathrm{X_1} = \mathrm{true}, \mathrm{X_2} = \mathrm{false})$, we access the corresponding term of the network polynomial by setting indicators x_1 and $\bar{x}_2$ to one and the rest to zero. To find $P(\mathrm{X_2} = \mathrm{true})$, we fix evidence on $\mathrm{X_2}$ by setting x_2 to one and $\bar{x}_2$ to zero and marginalize $\mathrm{X_1}$ by setting both x_1 and $\bar{x}_1$ to one. Notice that there are two reasons we might set an indicator $x_i = 1$: *(1)* evidence $\{\mathrm{X_i} = \mathrm{true}\}$, in which case we set $\bar{x}_i = 0$ and *(2)* marginalization of $\mathrm{X_i}$, where $\bar{x}_i = 1$ as well. In general the role of an indicator x_i is to determine whether terms compatible with variable state $\mathrm{X_i} = \mathrm{true}$ are included in the summation, and similarly for $\bar{x}_i$.

With this notation, the partition function Z can be computed by setting all indicators of all variables to one.

The network polynomial has size exponential in the number of variables, but in many cases it can be represented more compactly using a sum-product network [120, 38].

Definition 1. *(Poon & Domingos, 2011) A sum-product network (SPN) over variables*

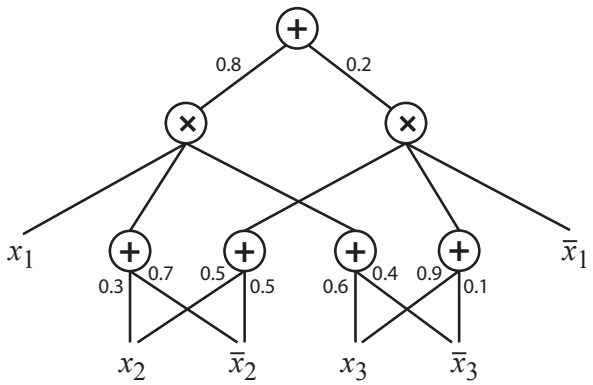

Figure 3.1: SPN over Boolean variables X_1, X_2, X_3

$\mathrm{X}_1, \ldots, \mathrm{X}_\mathrm{d}$ *is a rooted directed acyclic graph whose leaves are the indicators* $x_1, \ldots, x_d$ *and* $\bar{x}_1, \ldots, \bar{x}_d$ *and whose internal nodes are sums and products. Each edge* (i, j) *emanating from a sum node* i *has a non-negative weight* w_{ij}*. The value of a product node is the product of the values of its children. The value of a sum node is* $\sum_{j \in Ch(i)} w_{ij} v_j$*, where* $Ch(i)$ *are the children of* i *and* v_j *is the value of node* j*. The value of an SPN* $S[x_1, \bar{x}_1, \ldots, x_d, \bar{x}_d]$ *is the value of its root.*

If we could replace the exponential sum over variable states in the partition function with the linear evaluation of the network, inference would be tractable. For example, the SPN in Figure 1 represents the joint probability of three Boolean variables $P(\mathrm{X}_1, \mathrm{X}_2, \mathrm{X}_3)$ in the Bayesian network $\mathrm{X}_2 \leftarrow \mathrm{X}_1 \rightarrow \mathrm{X}_3$ using six indicators $S[x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3]$. To compute $P(\mathrm{X}_1 = \text{true})$, we could sum over the joint states of X_2 and X_3, evaluating the network a total of four times $S[1, 0, 0, 1, 0, 1] + \ldots + S[1, 0, 1, 0, 1, 0]$. Instead, we set the indicators so that the network sums out both X_2 and X_3. An indicator setting of S[1,0,1,1,1,1] computes the sum over all states compatible with our evidence $\mathrm{e} = \{\mathrm{X}_1 = \text{true}\}$ and requires only one evaluation.

However, not every SPN will have this property. If a linear evaluation of an SPN with indicators set to represent evidence equals the exponential sum over all variable states consistent with that evidence, the SPN is *valid*.

Definition 2. *(Poon & Domingos, 2011) A sum-product network S is* valid *iff $S(\mathrm{e}) = \Phi_S(\mathrm{e})$ for all evidence* e.

In their paper, Poon and Domingos prove that there are two conditions sufficient for validity: completeness and consistency.

Definition 3. *(Poon & Domingos, 2011) A sum-product network is* complete *iff all children of the same sum node have the same scope.*

Definition 4. *(Poon & Domingos, 2011) A sum-product network is* consistent *iff no variable appears negated in one child of a product node and non-negated in another.*

Theorem 2. *(Poon & Domingos, 2011) A sum-product network is valid if it is complete and consistent.*

The scope of a node is defined as the set of variables that have indicators among the node's descendants. To "appear in a child" means to be among that child's descendants. If a sum node is incomplete, the SPN will *undercount* the true marginals. Since an incomplete sum node has scope larger than a child, that child will be non-zero for more than one state of the sum (e.g. if $S[x_1, \bar{x}_1, x_2, \bar{x}_2] = (x_1 + x_2)$, $S[1, 0, 1, 1] < S[1, 0, 1, 0] + S[1, 0, 0, 1]$). If a product node is inconsistent, the SPN will *overcount* the marginals as it will incorporate impossible states (e.g. $x_1 \times \bar{x}_1$) into its computation.

Poon and Domingos show how to generatively train the parameters of an SPN. One method is to compute the likelihood gradient and optimize with gradient descent (GD). They also show how to use expectation maximization (EM) by considering each sum node as the marginalization of a hidden variable [44]. They found that online EM using most probable explanation (MPE or "hard") inference worked the best for their image completion task.

Gradient diffusion is a key issue in training deep models. It is commonly observed in neural networks that when the gradient is propagated to lower layers it becomes less informative [9]. When every node in the network takes fractional responsibility for the errors

of a top level node, it becomes difficult to steer parameters out of local minima. Poon and Domingos also saw this effect when using gradient descent and EM to train SPNs. They found that online hard EM could provide a sparse but strong learning signal to synchronize the efforts of upper and lower nodes. Note that hard training is not exclusive to EM. In the next section we show how to discriminatively train SPNs with hard gradient descent.

3.3 Discriminative Learning of SPNs

We define an SPN $S[\boldsymbol{y}, \boldsymbol{h}|\boldsymbol{x}]$ that takes as input three disjoint sets of variables $\mathbf{H}$, $\mathbf{Y}$, and $\mathbf{X}$ (hidden, query, and given). We denote the setting of all $\boldsymbol{h}$ indicator functions to 1 as $S[\boldsymbol{y}, \mathbf{1}|\boldsymbol{x}]$, where the bold $\mathbf{1}$ is a vector. We do not sum over states of given variables $\mathbf{X}$ when discriminatively training SPNs. Given an instance, we treat $\mathbf{X}$ as constants. This means that one ignores $\mathbf{X}$ variables in the scope of a node when considering completeness and consistency. Since adding a constant as a child to a product node cannot make that product inconsistent, a variable x can be the child of any product node in a valid SPN. To maintain completeness, x can only be the child of a sum node that has scope outside of $\mathbf{Y}$ or $\mathbf{H}$.

The parameters of an SPN can be learned using an online procedure as in Algorithm 1 as proposed by Poon and Domingos. The three dimensions of the algorithm are generative vs. discriminative, the inference procedure, and the weight update. Poon and Domingos discussed generative gradient descent with marginal inference as well as EM with marginal and MPE inference. In this section we will derive discriminative gradient descent with

Algorithm 1: Learn SPN parameters

Input: Set D of instances over variables $\mathbf{X}$ and label variables $\mathbf{Y}$, a valid SPN S with initialized parameters.
Output: An SPN with learned weights
repeat
  forall the $d \in D$ **do**
    UpdateWeights(S, Inference(S,$\mathbf{x_d}$,$\mathbf{y_d}$))
until *convergence or early stopping condition*;

marginal and MPE inference, where hard gradient descent can also be used for generative training. EM is not typically used for discriminative training as it requires modification to lower bound the conditional likelihood [132] and there may not be a closed form for the M-step.

3.3.1 Discriminative Training with Marginal Inference

A component of the gradient of the conditional log likelihood takes the form

$$\begin{aligned}\frac{\partial}{\partial w}\log P(\mathbf{y}|\mathbf{x}) &= \frac{\partial}{\partial w}\log\sum_{\mathbf{h}}\Phi(\mathbf{Y}=\mathbf{y},\mathbf{H}=\mathbf{h}|\mathbf{x}) - \frac{\partial}{\partial w}\log\sum_{\mathbf{y}',\mathbf{h}}\Phi(\mathbf{Y}=\mathbf{y}',\mathbf{H}=\mathbf{h}|\mathbf{x}) \\ &= \frac{1}{S[\boldsymbol{y},\mathbf{1}|\boldsymbol{x}]}\frac{\partial S[\boldsymbol{y},\mathbf{1}|\boldsymbol{x}]}{\partial w} - \frac{1}{S[\mathbf{1},\mathbf{1}|\boldsymbol{x}]}\frac{\partial S[\mathbf{1},\mathbf{1}|\boldsymbol{x}]}{\partial w}\end{aligned}$$

where the two summations are separate bottom-up evaluations of the SPN with indicators set as $S[\boldsymbol{y},\mathbf{1}|\boldsymbol{x}]$ and $S[\mathbf{1},\mathbf{1}|\boldsymbol{x}]$, respectively.

The partial derivatives of the SPN with respect to all weights can be computed with backpropagation, detailed in Algorithm 2. After performing a bottom-up evaluation of the SPN, partial derivatives are passed from parent to child as follows from the chain rule and described in [39]. The form of backpropagation presented takes time linear in the number of nodes in the SPN if product nodes have a bounded number of children.

Our gradient descent update then follows the direction of the partial derivative of the conditional log likelihood with learning rate η: $\Delta w = \eta\frac{\partial}{\partial w}\log P(\mathbf{y}|\mathbf{x})$. After each gradient step we optionally renormalize the weights of a sum node so they sum to one. Empirically we have found this to produce the best results. The second SPN evaluation that marginalizes $\mathbf{H}$ and $\mathbf{Y}$ can reuse computation from the first, for example, when $\mathbf{Y}$ is modeled by a root sum node. In this case the values of all non-root nodes are equivalent between the two evaluations. For any architecture, one can memoize values of nodes that do not have a query variable indicator as a descendant.

Algorithm 2: SPN Backprop

Input: A valid SPN S, where S_n denotes the value of node n after bottom-up evaluation.

Output: Partial derivatives of the SPN with respect to every node $\frac{\partial S}{\partial S_n}$ and weight $\frac{\partial S}{\partial w_{i,j}}$

Initialize all $\frac{\partial S}{\partial S_n} = 0$ except $\frac{\partial S}{\partial S} = 1$

forall the $n \in S$ *in top-down order* **do**

  if *n is a sum node* **then**

    forall the $j \in Ch(n)$ **do**

      $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + w_{n,j} \frac{\partial S}{\partial S_n}$

      $\frac{\partial S}{\partial w_{n,j}} \leftarrow S_j \frac{\partial S}{\partial S_n}$

  else

    forall the $j \in Ch(n)$ **do**

      $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + \frac{\partial S}{\partial S_n} \prod_{k \in Ch(n) \setminus \{j\}} S_k$

3.3.2 Discriminative Training with MPE Inference

There are several reasons why MPE inference is appealing for discriminatively training SPNs. As discussed above, hard inference was crucial for overcoming gradient diffusion when generatively training SPNs. For many applications the goal is to predict the most probable structure, and therefore it makes sense to use this also during training. Finally, it is common to approximate summations with maximizations for reasons of speed or tractability. Though summation in SPNs is fast and exact, MPE inference is still faster. We derive discriminative gradient descent using MPE inference.

We define a max-product network (MPN) $M[\boldsymbol{y}, \boldsymbol{h}|\boldsymbol{x}]$ based on the max-product semiring. This network compactly represents the maximizer polynomial $\max_{\mathbf{x}} \Phi(\mathbf{x}) \prod(\mathbf{x})$, which computes the MPE [39]. To convert an SPN to an MPN, we replace each sum node by a max node, where weights on children are retained. The gradient of the conditional log likelihood with MPE inference is then

$$\frac{\partial}{\partial w} \log \tilde{P}(\mathbf{y}|\mathbf{x}) \quad = \quad \frac{\partial}{\partial w} \log \max_{\mathbf{h}} \Phi(\mathbf{Y} = \mathbf{y}, \mathbf{H} = \mathbf{h}|\mathbf{x}) - \frac{\partial}{\partial w} \log \max_{\mathbf{y}', \mathbf{h}} \Phi(\mathbf{Y} = \mathbf{y}', \mathbf{H} = \mathbf{h}|\mathbf{x})$$

where the two maximizations are computed by $M[\boldsymbol{y}, \mathbf{1}|\boldsymbol{x}]$ and $M[\mathbf{1}, \mathbf{1}|\boldsymbol{x}]$. MPE inference also consists of a bottom-up evaluation followed by a top-down pass. Inference yields a branching path through the SPN called a complete subcircuit that includes an indicator (and therefore assignment) for every variable [39]. Analogous to Viterbi decoding, the path starts at the root node and at each max (formerly sum) node it only travels to the max-valued child. At product nodes, the path branches to all children. We define W as the multiset of weights traversed by this path[1]. The value of the MPN takes the form of a product $\prod_{w_i \in W} w_i^{c_i}$, where c_i is the number of times w_i appears in W. The partial derivatives of the MPN with respect to all nodes and weights is computed by Algorithm 2 modified to accommodate MPNs: *(1)* S becomes M, *(2)* when n is a sum node, the body of the *forall* loop is run once for j as the max-valued child.

The partial derivative of the logarithm of an MPN with respect to a weight takes the form

$$\frac{\partial \log M}{\partial w_i} = \frac{\partial \log M}{\partial M}\frac{\partial M}{\partial w_i} = \frac{1}{M}\frac{\partial M}{\partial w_i} = \frac{c_i \cdot w_i^{c_i - 1}\prod_{w_j \in W \setminus \{w_i\}} w_j^{c_j}}{\prod_{w_j \in W} w_j^{c_j}} \quad = \quad \frac{c_i}{w_i}$$

The gradient of the conditional log likelihood with MPE inference is therefore $\Delta c_i / w_i$, where $\Delta c_i = c_i' - c_i''$ is the difference between the number of times w_i is traversed by the two MPE inference paths in $M[\boldsymbol{y}, \mathbf{1}|\boldsymbol{x}]$ and $M[\mathbf{1}, \mathbf{1}|\boldsymbol{x}]$, respectively. The hard gradient update is then $\Delta w_i = \eta \frac{\partial}{\partial w_i} \log \tilde{P}(\mathbf{y}|\mathbf{x}) = \eta \frac{\Delta c_i}{w_i}$.

The hard gradient for a training instance $(\mathbf{x_d}, \mathbf{y_d})$ is illustrated in Figure 2. In the first two expressions, the complete subcircuit traveled by each MPE inference is shown in bold. Product nodes do not have weighted children, so they do not appear in the gradient, depicted in the last expression

We can also easily add regularization to SPN training. An L2 weight penalty takes the familiar form of $-\lambda||\mathbf{w}||^2$ and partial derivatives $-2\lambda w_i$ can be added to the gradient. With

[1] A consistent SPN allows for MPE inference to reach the same indicator more than once in the same branching path

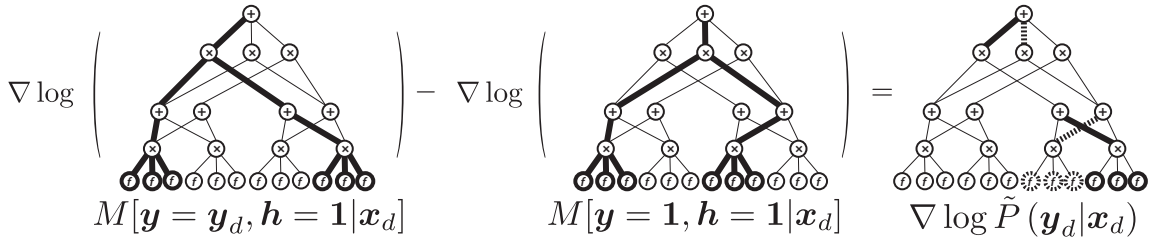

Figure 3.2: Positive and negative terms in the hard gradient. The root node sums out the variable Y, the two sum nodes on the left sum out the hidden variable H_1, the two sum nodes on the right sum out H_2, and a circled 'f' denotes an input variable X_i. Dashed lines indicate negative elements in the gradient.

an appropriate optimization method, an L1 penalty could also be used for learning with marginal inference on dense SPN architectures. However, sparsity is not as important for SPNs as it is for Markov random fields, where a non-zero weight can have outsize impact on inference time; with SPNs inference is always linear with respect to model size.

A summary of the variations of Algorithm 1 is provided in Tables 3.1 and 3.2. The generative hard gradient can be used in place of online EM for datasets where it would be prohibitive to store inference results from the past epoch. For architectures that have high fan-in sum nodes, soft inference may be able to separate groups of modes faster than hard inference, which can only alter one child of a sum node at a time.

We observe the similarity between the updates of hard EM and hard gradient descent. In particular, if we reparameterize the SPN so that each child of a sum node is weighted by $w_i = e^{w'_i}$, the form of the partial derivative of the log MPN becomes

$$\frac{\partial \log M}{\partial w'_i} = \frac{1}{M}\frac{\partial M}{\partial w'_i} = \frac{c_i \prod_{w'_j \in W'} e^{c_j \cdot w'_j}}{\prod_{w'_j \in W'} e^{c_j \cdot w'_j}} = c_i$$

This means that the hard gradient update for weights in logspace is $\Delta w'_i = \Delta c_i$, which resembles structured perceptron [32].

Table 3.1: Inference procedures

Node	Soft Inference	Hard Inference
Sum	$\frac{\partial S}{\partial S_n} = \sum_{k \in Pa(n)} \frac{\partial S}{\partial S_k} \prod_{l \in Ch(k) \setminus \{n\}} S_l$	$\frac{\partial M}{\partial M_n} = \sum_{k \in Pa(n)} \frac{\partial M}{\partial M_k} \prod_{l \in Ch(k) \setminus \{n\}} M_l$
Product	$\frac{\partial S}{\partial S_n} = \sum_{k \in Pa(n)} w_{kn} \frac{\partial S}{\partial S_k}$	$\frac{\partial M}{\partial M_n} = \sum_{k \in Pa(n)} \begin{cases} w_{kn} \frac{\partial M}{\partial M_k} & : w_{kn} \in W \\ 0 & : \text{otherwise} \end{cases}$
Weight	$\frac{\partial S}{\partial w_{ki}} = \frac{\partial S}{\partial S_k} S_i$	$\frac{\partial M}{\partial w_{ki}} = \frac{\partial M}{\partial M_k} M_i$

Table 3.2: Weight updates

Update	Soft Inference	Hard Inference
Gen. GD	$\Delta w = \eta \frac{\partial S[\boldsymbol{x},\boldsymbol{y}]}{\partial w}$	$\Delta w_i = \eta \frac{c_i}{w_i}$
Gen. EM	$P(\mathrm{H_k} = \mathrm{i} \vert \mathbf{x}, \mathbf{y}) \propto w_{ki} \frac{\partial S[\boldsymbol{x},\boldsymbol{y}]}{\partial S_k} S_i$	$P(\mathrm{H_k} = \mathrm{i} \vert \mathbf{x}, \mathbf{y}) = \begin{cases} 1 & : w_{ki} \in W \\ 0 & : \text{otherwise} \end{cases}$
Disc. GD	$\Delta w = \eta \left(\frac{1}{S[\boldsymbol{y},\mathbf{1} \vert \boldsymbol{x}]} \frac{\partial S[\boldsymbol{y},\mathbf{1} \vert \boldsymbol{x}]}{\partial w} - \frac{1}{S[\mathbf{1},\mathbf{1} \vert \boldsymbol{x}]} \frac{\partial S[\mathbf{1},\mathbf{1} \vert \boldsymbol{x}]}{\partial w} \right)$	$\Delta w_i = \eta \frac{\Delta c_i}{w_i}$

3.4 Experiments

We have applied discriminative training of SPNs to image classification benchmarks. CIFAR-10 and STL-10 are standard datasets for deep networks and unsupervised feature learning. Both are 10-class small image datasets.

We follow the feature extraction pipeline of Coates et al. [31], which was also used to learn pooling functions [76]. The procedure consists of extracting 4×10^5 6x6 pixel patches from the training set images, ZCA whitening those patches [73], running k-means for 50 rounds, and then normalizing the dictionary to have zero mean and unit variance. We then use the dictionary to extract K features at every 6x6 pixel site in the image (unit stride) with the "triangle" encoding $f_k(x) = \max\{0, \bar{z} - z_k\}$, where $z_k = ||x - c_k||_2$, c_k is the k-th item in the dictionary, and $\bar{z}$ is the average z_k. For each image of CIFAR-10, for example, this yields a $27 \times 27 \times K$ feature vector that is finally downsampled by max-pooling to a $G \times G \times K$ feature vector.

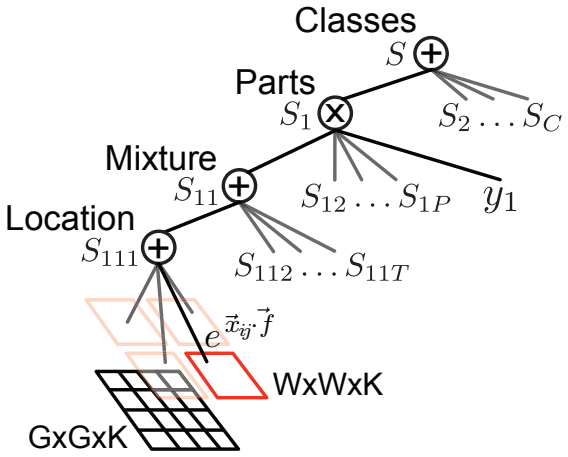

Figure 3.3: SPN architecture for experiments. Hidden variable indicators omitted for legibility.

We experiment with a simple architecture that allows for discriminative learning of local structure. This architecture cannot be generatively trained as it violates consistency over $\mathbf{X}$. Inspired by the successful star models in Felzenszwalb et al. [50], we construct a network with C classes, P parts per class, and T mixture components per part. A part is a pattern of image patch features that can occur anywhere in the image (e.g. an arrangement of patches that defines a curve). Each part filter $\vec{f}_{cpt}$ is of dimension $W \times W \times K$ and is initialized to $\vec{0}$. The root of the SPN is a sum node with a child S_c for each class c in the dataset multiplied by the indicator for that state of the label variable Y. S_c is a product over P nodes S_{cp}, where each S_{cp} is a sum node over T nodes S_{cpt}. The hidden variables $\mathbf{H}$ represent the choice of cluster in the mixture over a part and its position (S_{cp} and S_{cpt}, respectively). Finally, S_{cpt} sums over positions i, j in the image of the logistic function $e^{\vec{x}_{ij} \cdot \vec{f}_{cpt}}$ where the given variable $\vec{x}_{ij}$ is the same dimension as f and parts can overlap.

Notice that the mixture S_{cp} models an additional level of spatial structure on top of the image patch features learned by k-means. Coates and Ng [30] also learn higher-order structure, but whereas our method learns structure discriminatively in the context of a parts-based model, their unsupervised algorithm greedily groups features based on correlation and

is unable to learn mixtures. Compared with the pooling functions in Jia et al. [76] that model independent translation of patch features, our architecture models how nearby features move together. Other deep probabilistic architectures should be able to model high-level structure, but considering the difficulty in training these models with approximate inference, it is hard to make full use of their representational power. Unlike the star model of Felzenswalb et al. [50] that learns filters over predefined HOG image features, our SPN learns on top of learned image features that can model color and detailed patterns.

Generative SPN architectures on the same features produce unsatisfactory results as generative training is led astray by the large number of features, very few of which differentiate labels. In the generative SPN paper [120], continuous variables are modeled with univariate Gaussians at the leaves (viewed as a sum node with infinite children but finite weight sum). With discriminative training, $\mathbf{X}$ can be continuous because we always condition on it, which effectively folds it into the weights.

All networks are learned with stochastic gradient descent regularized by early stopping. We found that using marginal inference for the root node and MPE inference for the rest of the network worked best. This allows the SPN to continue learning the difference between classes even when it correctly classifies a training instance. The fraction of the training set reserved for validation with CIFAR-10 and STL-10 were 10% and 20%, respectively. Learning rates, P, and T were chosen based on validation set performance.

3.4.1 Results on CIFAR-10

CIFAR-10 consists of 32x32 pixel images: 5×10^4 for training and 10^4 for testing. We first compare discriminative SPNs with other methods as we vary the size of the dictionary K. The results are seen in Figure 4. To fairly compare with recent work [31, 76] we also set $G = 4$. In general, we observe that SPNs can achieve higher performance using half as many features as the next best approach, the learned pooling function. We hypothesize that this is because the SPN architecture allows us to discriminatively train large moveable parts, image structure that cannot be captured by larger dictionaries. In Jia et al. [76] the pooling

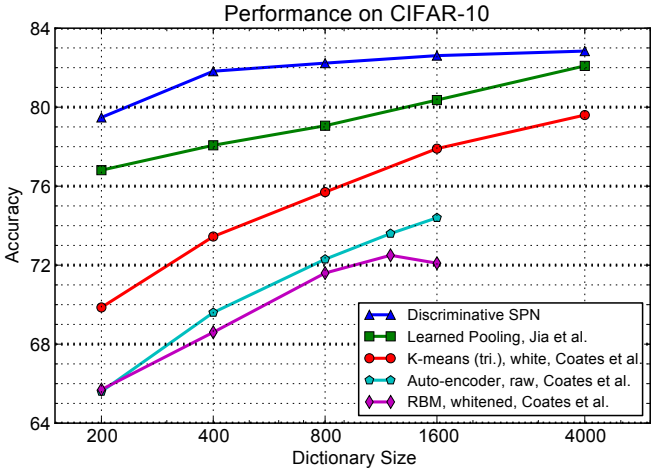

Figure 3.4: Impact of dictionary size K with a 4x4 pooling grid (W=3) on CIFAR-10 test accuracy

functions blur individual features (i.e. a 6x6 pixel dictionary item), from which the classifier may have trouble inferring the coordination of image parts.

We then experimented with a finer grid and fewer dictionary items ($G = 7$, $K = 400$). Pooling functions destroy information, so it is better if less is done before learning. Finer grids are less feasible for the method in Jia et al. [76] as the number of rectangular pooling functions grows $O(G^4)$. Our best test accuracy of 83.96% was achieved with $W = 3$, $P = 200$, and $T = 2$, chosen by validation set performance. At the time of publication, this architecture achieved the highest test accuracy on the CIFAR-10 dataset without data augmentation, remarkably using one fifth the number of features of the next best approach. We compare top CIFAR-10 results in Table 3.3, highlighting the dictionary size of systems that use the feature extraction from Coates et al. [31].

3.4.2 Results on STL-10

STL-10 has larger 96x96 pixel images and less labeled data (5,000 training and 8,000 test) than CIFAR-10 [31]. The training set is mapped to ten predefined folds of 1,000 images.

Table 3.3: Test accuracies on CIFAR-10.

Method	**Dictionary**	**Accuracy**
Logistic Regression [122]		36.0%
SVM [14]		39.5%
SIFT [14]		65.6%
mcRBM [122]		68.3%
mcRBM-DBN [122]		71.0%
Convolutional RBM [31]		78.9%
K-means (Triangle) [31]	4000, 4x4 grid	79.6 %
HKDES [15]		80.0%
3-Layer Learned RF [30]	1600, 9x9 grid	82.0%
Learned Pooling [76]	6000, 4x4 grid	83.11%
Discriminative SPN	**400, 7x7 grid**	**83.96%**
Multi-Column DNNs [28]		88.79%
Maxout Networks [60]		90.65%
Deep Residual Networks [65]		93.57%
All Convolutional Net [144]		95.59%
Fractional Max-Pooling [61]		96.53%

We experimented on the STL-10 dataset in a manner similar to CIFAR-10, ignoring the 10^5 items of unlabeled data. Ten models were trained on the pre-specified folds, and test accuracy is reported as an average. With K=1600, G=8, W=4, P=10, and T=3 we achieved 62.3% ($\pm$ 1.0% standard deviation among folds), the highest published test accuracy as of writing. Notably, this includes approaches that make use of the unlabeled training images. Like Coates and Ng [30], our architecture learns local relations among different feature maps. However, the SPN is able to discriminatively learn latent mixtures, which can encode a more nuanced decision boundary than the linear classifier used in their work. After we carried out our experiments, Bo et al. [16] reported a higher accuracy with their unsupervised features and a linear SVM. Just as with the features of Coates et al. [31], we anticipate that using an SPN instead of the SVM would be beneficial by learning spatial structure that the SVM cannot model.

Table 3.4: Comparison of average test accuracies on all folds of STL-10.

Method	Accuracy ($\pm\sigma$)
1-layer Vector Quantization [29]	54.9% ($\pm$ 0.4%)
1-layer Sparse Coding [29]	59.0% ($\pm$ 0.8%)
3-layer Learned Receptive Field [30]	60.1% ($\pm$ 1.0%)
Discriminative SPN	**62.3% ($\pm$ 1.0%)**
Hierarchical Matching Pursuit [16]	64.5% ($\pm$ 1.0%)
Multi-Task Bayesian Optimization [147]	70.1% ($\pm$ 0.6%)
Stacked What-Where Auto-encoders [167]	74.3%
Exemplar Convnets [47]	75.4% ($\pm$ 0.3%)

3.4.3 Contemporary Results

Tables 3.3 and 3.4 show that after publication of this work, several methods have outperformed SPNs on these datasets. These papers largely agree on several techniques to achieve state of the art results. One salient trend for neural network architectures is the use of many convolutional layers with very small kernels (3×3 and smaller). This simultaneously achieves the benefits of increased depth through non-linear functions and reduces the number of parameters in the model. Another key factor in these approaches is extensive augmentation of the training set with affine distortions and color shifts. Some of these papers average test-time predictions from an ensemble either of different models trained with different distortions [28] or copies of the same model viewing the test image with different distortions [61]. Applying these insights to discriminatively-trained SPNs could significantly improve their performance. However, the optimization of a deep network using an augmented training set is expensive and complicated. Chapters 5 and 6 develop techniques to address these weaknesses.

3.5 Conclusion

In this chapter we introduced the first algorithms for learning SPNs discriminatively, using a form of backpropagation to compute gradients. Discriminative training allows for a wider

variety of SPN architectures than generative training because completeness and consistency do not have to be maintained over evidence variables. We proposed both "soft" and "hard" gradient algorithms, using marginal inference in the "soft" case and MPE inference in the "hard" case. The latter successfully combats the diffusion problem, allowing deep networks to be learned. We presented experiments on image classification benchmarks that show discriminative SPNs can outperform other deep models.

Chapter 4

LEARNING THE STRUCTURE OF SUM-PRODUCT NETWORKS

4.1 Introduction

As shown in the previous chapter, the parameters of a given SPN structure can be learned generatively or discriminatively. These methods have performed well on several datasets, but they used pre-defined SPN structures that were both expensive to learn (because of a large number of nodes) and insufficiently flexible (because the best nodes were often not in the predefined structure). This trade-off between cost of learning and flexibility can be avoided, or at least ameliorated, by learning the structure of the SPN from data.

In this chapter, we propose the first algorithm for learning the structure of SPNs that does not sacrifice any of their expressiveness. This work introduced a simplified definition of SPNs, used in the background of this dissertation, that incorporates Poon and Domingos' conditions for tractability, avoids the use of indicator functions and network polynomials, and generalizes more easily to continuous variables. We begin the chapter by presenting our algorithm, which takes advantage of this simplification by having a recursive structure that parallels the recursive structure of the definition. It can be viewed as an intimate combination of mixture model EM (for learning sum nodes) and graphical model structure learning (for learning product nodes). We then review related structure learning algorithms. Finally, we test our algorithm on a large number of datasets from a wide variety of domains. Surprisingly, the learned SPNs typically have comparable likelihoods to unrestricted graphical models learned on the same data. At inference time they dominate in both speed and accuracy.

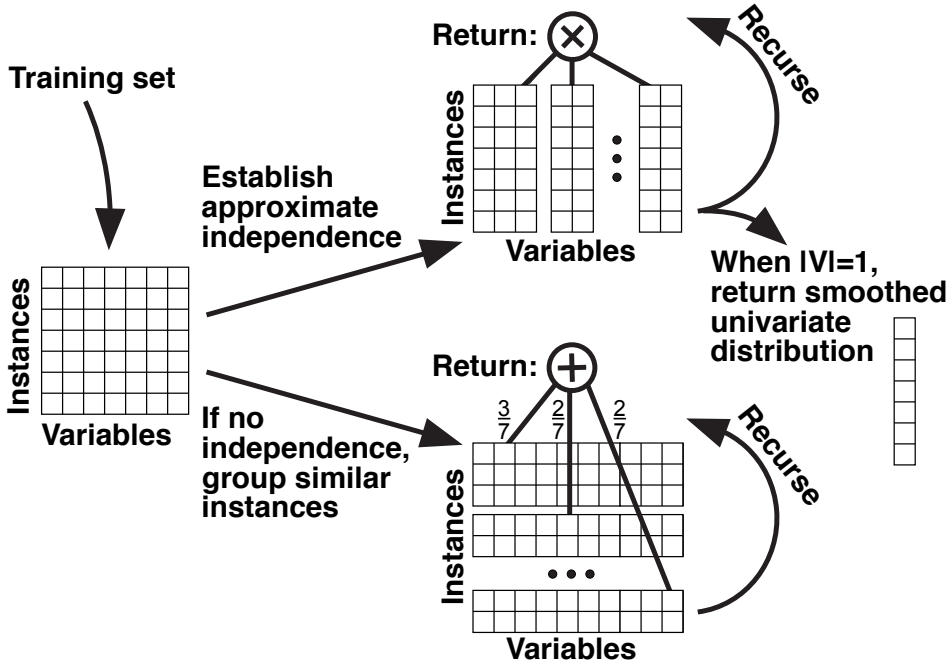

Figure 4.1: A recursive algorithm for learning SPNs.

4.2 Structure Learning

The method we propose for learning SPN structure is summarized in Algorithm 3 and illustrated in Figure 4.1. LearnSPN inputs an i.i.d. sample of a vector-valued variable, in the form of a matrix of instances by variables. If the vector is of unit length, LearnSPN returns the corresponding univariate distribution, with MAP estimates of the parameters. For example, for discrete variables the distribution may be a multinomial with Dirichlet prior, and for continuous ones it may be a normal with normal-Wishart prior. If the vector is of length greater than one, LearnSPN recurses on submatrices with either fewer rows or fewer columns. If it is able to split the variables into mutually independent subsets, it recurses on those, and returns the product of the resulting SPNs. Otherwise, it clusters the instances into similar subsets, recurses on those, and returns the weighted sum of the resulting SPNs. The weight of an SPN is the fraction of instances in the corresponding subset; it can also be smoothed using a Dirichlet prior.

Algorithm 3: LearnSPN(T, V)

Input: set of instances T and set of variables V
Output: an SPN representing a distribution over V learned from T
if $|V| = 1$ **then**
  return *univariate distribution estimated from the variable's values in* T
else
  partition V into approximately independent subsets V_j
  if *success* **then**
    return $\prod_j$ *LearnSPN*(T, V_j)
  else
    partition T into subsets of similar instances T_i
    return $\sum_i \frac{|T_i|}{|T|} \cdot$ *LearnSPN*(T_i, V)

If there are no detectable dependencies among the variables, LearnSPN returns a fully factorized distribution. At the other extreme, if LearnSPN always fails to find independent subsets of variables until $|T| = 1$ (at which point all variables are independent), it returns a kernel density estimate of the distribution [111]. More typically, if $|T| \gg |V|$ LearnSPN will likely split on subsets of instances, and if $|V| \gg |T|$ it will likely split on subsets of variables. Crucially, LearnSPN can choose different variable splits for different sets of instances, resulting in tractable models with few or no conditional independences.

LearnSPN is an algorithm schema rather than a single algorithm. It can incorporate a variety of methods for splitting variables and instances into subsets. In particular, instances can be clustered using the EM algorithm [44], and this is the method we will use in the rest of this paper. At each splitting step, we assume a naive Bayes mixture model, where all variables are independent conditioned on the cluster: $P(V) = \sum_i P(C_i) \prod_j P(X_j|C_i)$, where C_i is the ith cluster and X_j is the jth variable. For soft EM, where instances can be fractionally assigned to clusters, T needs to be extended with a weight for each instance, and each instance is passed to each cluster it has nonzero weight in. However, this is considerably less efficient than hard EM, where each instance is wholly assigned to its most probable cluster, and we will use the latter method. In either case, the learned SPN weights are now the

mixing proportions $P(C_i)$. We use online EM with restarts, which automatically determines the number of clusters by assigning each new instance to its most likely cluster, possibly a new one. Overfitting is avoided via an exponential prior on the number of clusters.

Under this scheme, we can see that at each instance splitting step LearnSPN(T, V) locally maximizes the posterior probability of the sub-SPN over (T, V). The mixture model's posterior is also a lower bound on the posterior of the SPN that LearnSPN will ultimately return for (T, V), since the posterior can only increase when the recursive calls attempt to model dependencies between variables within each cluster.

An alternative to clustering instances is to split them according to the value of a specific variable or subset of variables. The best variables to split on can be chosen using a mutual information criterion. This results in an algorithm similar to Gogate et al.'s (2010) and with some of the flavor of learning graphical models with context-specific independence.

Variable splits can also be found in a number of ways. Since mutual information is a submodular function, Queyranne's algorithm can be used to find in cubic time a split of the variables into two subsets with minimum empirical mutual information [121, 24]. However, we have found this to be too slow in practice. Alternatively, we can consider only pairwise dependencies. In this case, we can apply an independence test to each pair of variables, form a graph with an edge between each pair of variables found to be dependent, and recurse on each connected component. If the graph has only one connected component, the variable split fails, and LearnSPN proceeds to form an instance split.

Let an *independence oracle* be an independence test that declares two variables X_1 and X_2 to be independent iff all subsets of variables $V_1 \ni X_1$ and $V_2 \ni X_2$ are independent. Using such an oracle, factorizing the sub-SPN into the connected components found causes no loss of likelihood. Let a *granularity sequence* be a choice of the number of clusters at each step of LearnSPN, and assume LearnSPN uses soft EM for instance clustering and maximum likelihood estimates for univariate distributions. Then LearnSPN returns a locally optimal SPN, in the sense that no higher-likelihood SPN can be reached from it by a local repartition of variables or instances. This can be summarized in the following proposition.

Table 4.1: Dataset statistics.

Data set	$\|V\|$	Train	Valid.	Test	Density
NLTCS	16	16181	2157	3236	0.332
MSNBC	17	291326	38843	58265	0.166
KDDCup 2k	65	180092	19907	34955	0.008
Plants	69	17412	2321	3482	0.180
Audio	100	15000	2000	3000	0.199
Jester	100	9000	1000	4116	0.608
Netflix	100	15000	2000	3000	0.541
Accidents	111	12758	1700	2551	0.291
Retail	135	22041	2938	4408	0.024
Pumsb-star	163	12262	1635	2452	0.270
DNA	180	1600	400	1186	0.253
Kosarak	190	33375	4450	6675	0.020
MSWeb	294	29441	3270	5000	0.010
Book	500	8700	1159	1739	0.016
EachMovie	500	4524	1002	591	0.059
WebKB	839	2803	558	838	0.064
Reuters-52	889	6532	1028	1540	0.036
20 Newsgrp.	910	11293	3764	3764	0.049
BBC	1058	1670	225	330	0.078
Ad	1556	2461	327	491	0.008

Table 4.2: Structure learning summary.

Data set	SPN LL	WinMine LL	SPN PLL	DP PLL	L1 PLL
NLTCS	-6.110	-6.025	-5.235	-4.941	-4.950
MSNBC	-6.113	**-6.041**	**-4.378**	-5.133	-6.060
KDDCup 2k	-2.182	-2.189	-2.084	-2.065	-2.067
Plants	-12.977	-12.647	-9.696	-9.636	**-9.405**
Audio	-40.503	-40.501	-38.604	-38.560	**-36.210**
Jester	-53.480	-53.847	-51.959	-53.048	**-48.754**
Netflix	-57.328	**-57.025**	-55.484	-56.419	**-51.067**
Accidents	-30.038	**-26.320**	-19.379	-26.766	**-12.446**
Retail	-11.043	-10.874	-10.590	-10.342	-10.326
Pumsb-star	-24.781	**-21.721**	-14.567	-23.665	**-9.653**
DNA	-82.523	**-80.646**	-63.431	-96.696	**-58.558**
Kosarak	-10.989	-10.834	-9.920	-10.579	-9.926
MSWeb	-10.252	**-9.697**	-8.917	-8.947	-8.728
Book	-35.886	-36.411	-34.153	-38.444	-36.346
EachMovie	-52.485	-54.368	-49.686	-64.808	-50.496
WebKB	-158.204	-157.433	-151.678	-174.448	-146.957
Reuters-52	-85.067	-87.555	-79.772	-104.173	-79.834
20 Newsgrp.	-155.925	-158.948	-151.887	-171.056	**-148.316**
BBC	-250.687	-257.861	**-245.340**	-272.916	-260.311
Ad	-19.733	-18.349	-11.326	-50.008	**-6.628**

Proposition 1. *Given a granularity sequence, LearnSPN with an independence oracle for variable splitting and EM for instance clustering returns a locally maximum likelihood SPN.*

4.3 Related Work

The first algorithm for SPN structure learning, proposed by Dennis and Ventura (2012), is quite limited. It essentially forms a set of hierarchical clusterings of the variables and builds the SPN based on them, and as a result is not able (except at the root) to take advantage of the context-specific independences that are crucial to SPNs' expressiveness. It clusters variables that have similar values in similar instances, and is therefore prone to splitting highly dependent variables, causing a large loss of likelihood. (For example, if X is the negation of Y, they will never be clustered.) The cost of learning and size of the SPN are worst-case exponential (order of the number of sum nodes raised to the number of sub-regions in a region of the SPN). The number of sum nodes in a region is a fixed input parameter, and not learnable. Weights are learned as a post-processing step, and cannot be optimized during structure learning. The algorithm is quite complex and *ad hoc*, with no guarantee of

Table 4.3: Average conditional log-likelihood normalized by number of query variables for two proportions of query and evidence. The last line shows the average time per query (ms/query).

	10% Query, 30% Evidence				50% Query, 30% Evidence			
Data	SPN	WinMine	DP	L1	SPN	WinMine	DP	L1
NLTCS	-0.232	-0.232	-0.231	-0.231	-0.371	-0.373	-0.374	-0.375
MSNBC	-0.236	**-0.234**	-0.237	-0.237	-0.337	-0.338	-0.341	-0.348
KDDCup 2k	-0.030	-0.030	-0.033	-0.089	**-0.032**	-0.036	-0.038	-0.077
Plants	-0.159	-0.158	-0.178	-0.174	**-0.161**	-0.226	-0.252	-0.246
Audio	-0.394	-0.401	-0.417	-0.397	**-0.390**	-0.780	-0.808	-0.773
Jester	-0.722	-0.638	-0.550	**-0.110**	-0.727	-0.910	-0.910	**-0.095**
Netflix	-0.573	-0.588	-0.612	-0.569	**-0.566**	-0.910	-0.910	-0.909
Accidents	**-0.300**	-0.526	-0.377	-0.564	**-0.245**	-0.790	-0.837	-0.805
Retail	-0.075	-0.076	-0.077	-0.638	**-0.078**	-0.130	-0.128	-0.496
Pumsb-star	**-0.151**	-0.364	-0.443	-0.480	**-0.125**	-0.553	-0.804	-0.648
DNA	**-0.486**	-0.675	-0.791	-0.702	**-0.444**	-0.820	-0.820	-0.820
Kosarak	**-0.055**	-0.057	-0.067	-0.424	**-0.053**	-0.111	-0.108	-0.297
MSWeb	**-0.036**	-0.038	-0.039	-0.040	**-0.033**	-0.079	-0.082	-0.080
Book	**-0.068**	-0.099	-0.105	-0.101	**-0.066**	-0.295	-0.344	-0.307
EachMovie	**-0.100**	-0.191	-0.228	-0.203	**-0.101**	-0.439	-0.704	-0.481
WebKB	**-0.182**	-0.504	-0.578	-0.518	**-0.184**	-0.708	-0.723	-0.723
Reuters-52	**-0.091**	-0.258	-0.307	-0.566	**-0.090**	-0.604	-0.722	-0.656
20 Newsgrp.	**-0.166**	-0.498	-0.532	-0.603	**-0.167**	-0.715	-0.722	-0.719
BBC	**-0.238**	-0.770	-0.789	-0.793	**-0.233**	-0.718	-0.718	-0.718
Ad	**-0.010**	-0.294	-0.126	-0.158	**-0.009**	-0.652	-0.518	-0.428
Time	24	1621	2397	5003	23	1629	2386	4858

finding even a local optimum of the likelihood, and no sense of how good the output SPN is. It has only been tested on an image completion task.

SPNs are related to multilinear formulas [124], arithmetic circuits [38], AND-OR graphs [41], and other compact representations. Lowd and Domingos [98] and Gogate et al. [59] proposed algorithms for learning tractable models that are related to SPNs but more restricted. Lowd and Domingos' algorithm learns a Bayesian network with context-specific independence using the network's inference cost as the regularization penalty. Gogate et al. learn tractable high-treewidth Markov networks by recursively searching for features that split the remaining variables into approximately independent subsets.

4.4 Experiments

We evaluated LearnSPN on twenty real-world datasets and compared with popular graphical model structure learning algorithms. This is a much greater number of datasets than is

typical in empirical evaluations of graphical model structure learning. The diverse set of domains includes click-through logs, plant habitats, nucleic acid sequences, collaborative filtering, and many others. The number of variables in a dataset ranges from 16 to 1556, and the number of instances varies from 2k to 388k.

We used the WinMine toolkit [26] to learn Bayesian network structure. WinMine allows context-specific independence in the form of a decision tree at each node [27], and is the most sophisticated graphical model structure learning package available. For Markov network structure learning, we ran algorithms by Della Pietra et al. (1997) and Ravikumar et al. (2010). The Della Pietra et al. algorithm is the canonical Markov network structure learner. Ravikumar et al. learns structure by inducing sparsity in a large set of weights with an L1 penalty.

The dimensions of the datasets are detailed in Table 4.1. We used discrete datasets because the three comparison systems do not support continuous variables. Thirteen of the datasets were processed by Lowd and Davis (2010); seven were assembled by Van Haaren and Davis (2012). We used the authors' train-validation-test splits, where most datasets reserve 10% of instances for validation and 15% for testing.

4.4.1 Learning

To cluster instances, we used hard incremental EM [108] over a naive Bayes mixture model with the exponential prior $P(S) \propto e^{-\lambda C|V|}$, where C is the number of clusters and λ is the cluster penalty. We ran ten restarts through the subset of instances T four times in random order. We estimated $P(X_j|C_i)$ with Laplace smoothing, adding 0.1 to each count.

For variable splits, we used a G-test of pairwise independence: $G(x_1, x_2) = 2 \sum_{x_1} \sum_{x_2} c(x_1, x_2) \cdot \log \frac{c(x_1,x_2) \cdot |T|}{c(x_1)c(x_2)}$, where the summations range over the values of each variable and $c(\cdot)$ counts the occurrences of a setting of a variable pair or singleton [166]. For each dataset, the cluster penalty λ and G-test significance p were chosen based on validation set performance[1].

[1]Grid search: $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$ and $p \in \{0.0015,\ 0.0001\}$.

We compared with Bayesian networks learned by the WinMine toolkit on test set log-likelihood (LL). We chose WinMine's per-parameter penalty κ according to validation set likelihood.

Since computing likelihood is intractable for the learned Markov networks, we instead compared test set pseudo-log-likelihood (PLL). This comparison greatly favors the Markov networks since they are trained to optimize PLL, and PLL is known to be a poor surrogate for likelihood. The methods of Della Pietra et al. (1997) and Ravikumar et al. (2010) are denoted as DP and L1, respectively. For DP, structure was learned using the open source code of Davis and Domingos (2010). L1 structure was learned using the OWL-QN package [4] for L1 logistic regression. Weights were learned by optimizing PLL with the limited-memory BFGS algorithm. Markov networks with learned structure and weights were provided by Van Haaren and Davis (2012).

Table 4.2 shows the log-likelihood and pseudo-log-likelihood of learned methods. In all tables, bold indicates p=0.05 significance on a paired t-test. For a majority of datasets, the SPN's likelihood is not significantly different from WinMine's likelihood. SPN PLL is also comparable to DP and L1 on more than half of the datasets, even though those systems have the advantage of directly optimizing it. Presumably, if the Markov networks' likelihoods could be measured, they would be systematically worse than the SPNs'.

LearnSPN's learning times ranged from 4m to 12h, WinMine's from 1s to 10m, DP's from 16m to 24h, and L1's from 9s to 6h. The current implementation of LearnSPN could be made faster in a number of ways, such as reusing counts, as is done in WinMine.

4.4.2 Inference

High likelihood is not very useful if approximate inference hurts accuracy at query time. In this section we test the speed and accuracy of the learned models at query time, where the goal is to infer the probability of a subset of the variables (the query) given the values of another (the evidence).

We generated queries from the test set of each dataset, varying the fraction of randomly

selected query and evidence variables. For each proportion of query and evidence variables (e.g., 10% query, 30% evidence), a thousand instances were randomly selected from the test set. For each selected test instance, a query $P(\mathbf{Q}{=}\mathbf{q}|\mathbf{E}{=}\mathbf{e})$ was created by randomly choosing the appropriate fraction of variables and assigning their values. We report the average conditional log-likelihood (CLL) of the queries given the evidence, which approximates the KL-divergence between the inferred probabilities and the true ones, sampling from the test set. We normalize the CLL by the number of query variables to facilitate comparison among datasets and query proportions.

The SPN performs exact inference in time linear in the number of edges. Since exact inference is intractable for the learned graphical models on these domains, we used Gibbs sampling as implemented by the open source Libra package [96]. We used the Libra default of 1000 samples with 100 iterations of burn-in.

As detailed in Table 4.3, SPNs are two orders of magnitude faster and significantly more accurate. We also ran Gibbs sampling with more chains and iterations; even when we allowed Gibbs sampling a factor of 10 or 100 more time, it did not close the gap in accuracy.

In Figure 4.2, for each of several representative datasets we plot graphs of normalized CLL where the fraction of query or evidence variables is varied. When query proportion is varied, evidence is fixed at 30%, and vice-versa. The learned SPNs have much higher CLL, a disparity that becomes more pronounced with longer queries and larger datasets.

To be sure that the difference between SPNs and the other methods was not just due to Gibbs sampling, we also tried using loopy belief propagation (BP). This only computes single-variable marginals, so we measured the conditional marginal log-likelihood (CMLL) instead of CLL: $CMLL(\mathbf{X}{=}\mathbf{x}|\mathbf{e}) = \sum_{x_i \in \mathbf{Q}} \log P(X_i{=}x_i|\mathbf{e})$, where $\mathbf{Q}$ is the set of query variables and $\mathbf{e}$ is the set of evidence variable values. We ran the same set of inference tasks as detailed above for the ten variable proportions presented in the graphs, running loopy BP[2] for the Bayesian and Markov networks and measuring CMLL for all models. Averaged across the

[2]We used Libra with its default of 50 maximum iterations and convergence threshold of 0.0001.

ten proportions, SPNs achieved higher CMLL than WinMine on 12 datasets, 18 compared with DP, and 17 compared with L1. This is remarkable, considering that the PLL training of Markov networks naturally pairs with testing CMLL.

Overall, these experiments show that learning SPNs is a very attractive alternative to learning graphical models. In learning accuracy, SPNs were comparable to Bayesian networks on most data sets, and comparable to or better than Markov networks. For inference accuracy and speed, SPNs are clearly the representation of choice. Further, inference in SPNs does not involve tuning settings or diagnosing convergence as with MCMC or BP, and an SPN predictably takes the same amount of time to compute any query.

4.5 Conclusion

In this chapter, we proposed a simple schema for learning SPN structure from data. Our algorithm recursively splits an SPN into a product of SPNs over independent sets of variables, if they can be found, or into a sum of SPNs learned from subsets of the instances, otherwise. In experiments on a large number of datasets, the SPNs obtained were typically comparable in likelihood to graphical models, but inference in them was much faster, and also more accurate. This work has been cited extensively since its publication in 2013, for example with improvements made to accuracy [125] and scalability [82]. The basic schema of the LearnSPN algorithm has far-reaching consequences for other domains. Drawing from this work, Friesen and Domingos [54] proposed the LearnSPF algorithm to learn tractable models on other semirings; for example, they learn structures on the min-sum semiring to accelerate optimization.

Code and supplemental results are available at http://spn.cs.washington.edu/learnspn/.

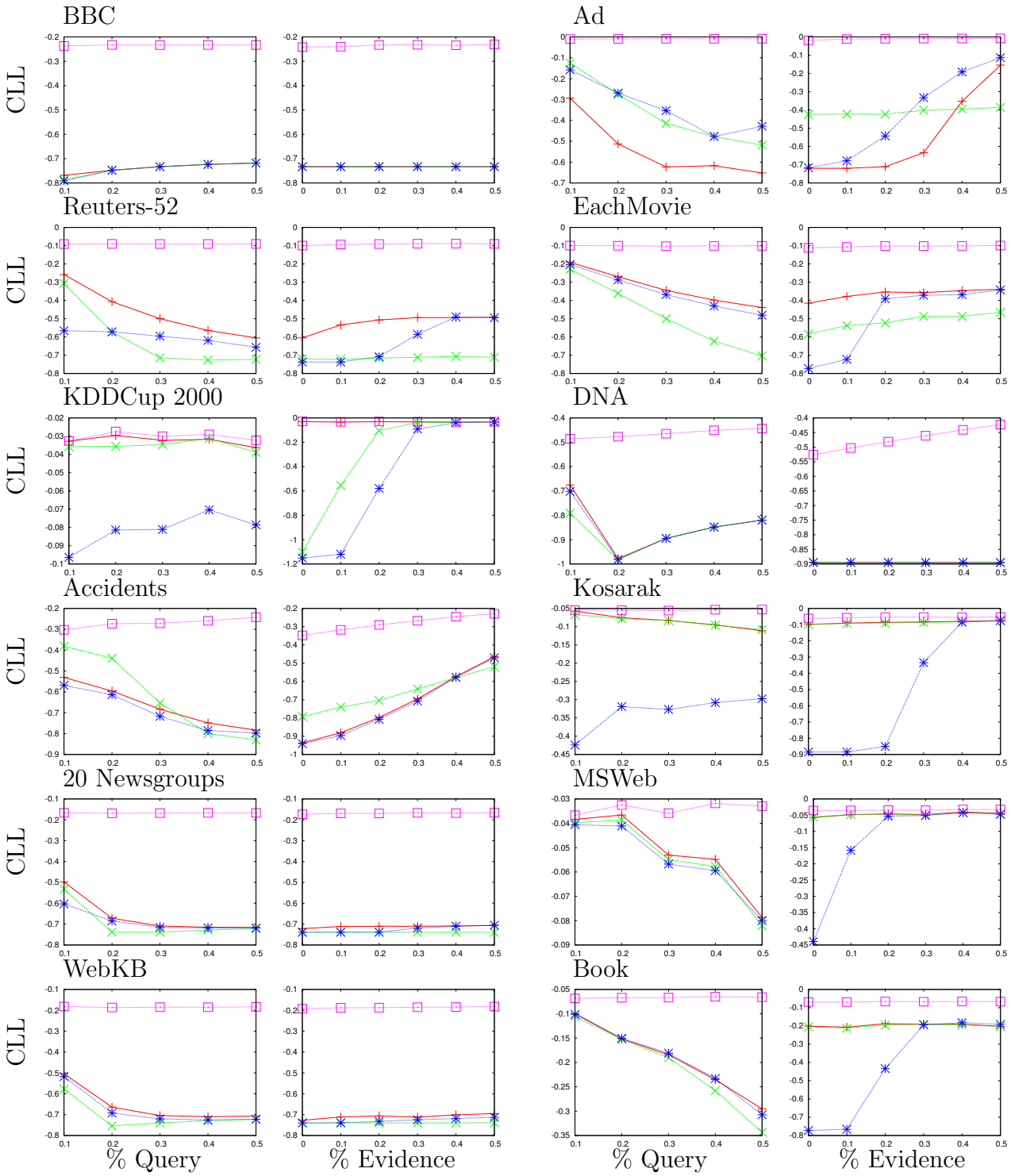

Figure 4.2: Average conditional log-likelihood normalized by number of query variables. Purple squares mark SPNs, blue asterisks L1, green Xs DP, and red crosses WinMine. For each dataset, the left chart fixes the fraction of evidence variables at 30% and varies the fraction of query variables; the right chart fixes query variables at 30% and varies evidence.

Chapter 5

DEEP SYMMETRY NETWORKS

5.1 Introduction

Learning a model over data is easier if its variables stay put from instance to instance. However, from the perspective of a computer, no two images are ever the same. Nuisance factors such as pose, lighting, part deformation, occlusion, and ultimately photon noise guarantee that pixel values will not match. It has been shown that if we could remove these factors, object recognition would be much easier [5, 95]. Convolutional neural networks (convnets) [91], the current state-of-the-art method for object recognition, capture only one type of invariance (translation); the rest have to be approximated via it and standard features. In practice, the best networks require enormous datasets which are further expanded by affine transformations [28, 84] yet are sensitive to imperceptible image perturbations [149]. In this chapter, we introduce deep symmetry networks, a generalization of convnets based on symmetry group theory [104] that makes it possible to capture a broad variety of invariances, and correspondingly improves generalization.

A symmetry group is a set of transformations that preserve the identity of an object and obey the group axioms. Most of the visual nuisance factors are symmetry groups themselves, and by incorporating them into our model we are able to reduce the sample complexity of learning from data transformed by these groups. Deep symmetry networks (symnets) form feature maps over any continuous symmetry group, rather than just the translation group. A feature map in a deep symmetry network is defined analogously to convnets as a filter that is applied at all points in the symmetry space. Each layer in our general architecture is constructed by applying every symmetry in the group to the input, computing features on the transformed input, and pooling over neighborhoods. The entire architecture is then

trained by backpropagation. The key challenge with extending convnets to high-dimensional symmetry spaces is that it is intractable to explicitly represent and compute with a high-dimensional feature map. We address this by approximating the map using kernel functions, which not only interpolate but also control pooling in the feature maps. In our experiments, we instantiate the architecture with the affine group, resulting in deep affine networks. In addition to translation, the affine group includes rotation, scaling and shear. The affine group of the two-dimensional plane is six-dimensional (i.e., an affine transformation can be represented by a point in 6D affine space). Compared to convnets, this architecture substantially reduces sample complexity on image datasets involving 2D and 3D transformations.

We begin the chapter with a review of symmetry group theory and its relation to sample complexity. We then describe symnets and their affine instance, and develop new methods to scale to high-dimensional symmetry spaces. We then detail related work from computer vision and machine learning. Finally, we present experiments on NORB and MNIST-rot show that affine symnets can reduce by a large factor the amount of data required to achieve a given accuracy level.

5.2 Symmetry Group Theory

A symmetry of an object is a transformation that leaves certain properties of that object intact [104]. A group is a set $\mathcal{S}$ with an operator $*$ on it with the four properties of closure, associativity, an identity element, and an inverse element. A symmetry group is a type of group where the group elements are functions and the operator is function composition. A simple geometric example is the symmetry group of a square, which consists of four reflections and $\{0, 1, 2, 3\}$ multiples of 90-degree rotations. These transformations can be composed together to yield one of the original eight symmetries. The identity element is the 0-degree rotation. Each symmetry has a corresponding inverse element. Composition of these symmetries is associative.

Lie groups are continuous symmetry groups whose elements form a smooth differentiable manifold. For example, the symmetries of a circle include reflections and rotations about the

center. The affine group is a set of transformations that preserves collinearity and parallel lines. The Euclidean group is a subgroup of the affine group that preserves distances, and includes the set of rigid body motions (translations and rotations) in three-dimensional space.

The elements of a symmetry group can be represented as matrices. In this form, function composition can be performed via matrix multiplication. The transformation $\mathbf{P}$ followed by $\mathbf{Q}$ (also denoted $\mathbf{Q} \circ \mathbf{P}$) is computed as $\mathbf{R} = \mathbf{QP}$. In this paper we treat the transformation matrix $\mathbf{P}$ as a point in D-dimensional space, where D depends on the particular representation of the symmetry group (e.g., $D = 6$ for affine transformations in the plane).

A generating set of a group is a subset of the group such that any group element can be expressed through combinations of generating set elements and their inverses. For example, a generating set of the translation symmetry group is $\{x \to x+\epsilon, y \to y+\epsilon\}$ for infinitesimal ϵ. We define the *k-neighborhood* of element f in group $\mathcal{S}$ under generating set $\mathcal{G}$ as the subset of $\mathcal{S}$ that can be expressed as f composed with elements of $\mathcal{G}$ or their inverses at most k times. With the previous example, the k-neighborhood of a translation vector f would take the shape of a diamond centered at f in the xy-plane.

The orbit of an object x is the set of objects obtained by applying each element of a symmetry group to x. Formally, a symmetry group $\mathcal{S}$ acting on a set of objects X defines an orbit for each $x \in X$: $O_x = \{s * x : s \in \mathcal{S}\}$. For example, the orbit of an image $I(u)$ whose points are transformed by the rotation symmetry group $s * I(u) = I(s^{-1} * u)$ is the set of images resulting from all rotations of that image. If two orbits share an element, they are the same orbit. In this way, a symmetry group $\mathcal{S}$ partitions the set of objects into unique orbits $X = \bigcup_a O_a$. If a data distribution $\mathcal{D}(x, y)$ has the property that all the elements of an orbit share the same label y, $\mathcal{S}$ imposes a constraint on the hypothesis class of a learner, effectively lowering its VC-dimension and sample complexity [2].

5.3 Deep Symmetry Networks

Deep symmetry networks represent rich compositional structure that incorporates invariance to high-dimensional symmetries. The ideas behind these networks are applicable to any

symmetry group, be it rigid-body transformations in 3D or permutation groups over strings. The architecture of a symnet consists of several layers of feature maps. Like convnets, these feature maps benefit from weight tying and pooling, and the whole network is trained with backpropagation. The maps and the filters they apply are in the dimension D of the chosen symmetry group $\mathcal{S}$.

A deep symmetry network has L layers $l \in \{1, ..., L\}$ each with I_l features and corresponding feature maps. A feature is the dot-product of a set of weights with a corresponding set of values from a local region of a lower layer followed by a nonlinearity. A feature map represents the application of a filter at all points in symmetry space. A feature at point $\mathbf{P}$ is computed from the feature maps of the lower layer at points in the k-neighborhood of $\mathbf{P}$. As $\mathbf{P}$ moves in the symmetry space of a feature map, so does its neighborhood of inputs in the lower layer. Feature map i of layer l is denoted $M[l, i] : \mathbb{R}^D \to \mathbb{R}$, a scalar function of the D-dimensional symmetry space. Given a generating set $\mathcal{G} \subset \mathcal{S}$, the points in the k-neighborhood of the identity element are stored in an array $\mathbf{T}[\,]$. Each filter i of layer l defines a weight vector $\mathbf{w}[l, i, j]$ for each point $\mathbf{T}[j]$ in the k-neighborhood. The vector $\mathbf{w}[l, i, j]$ is the size of I_{l-1}, the number of features in the underlying layer. For example, a feature in an affine symnet that detects a person would have positive weight for an arm sub-feature in the region of the k-neighborhood that would transform the arm relative to the person (e.g., smaller, rotated, and translated relative to the torso). The value of feature map i in layer l at point $\mathbf{P}$ is the dot-product of weights and underlying feature values in the neighborhood of $\mathbf{P}$ followed by a nonlinearity:

$$M[l, i](\mathbf{P}) = \sigma\left(v(\mathbf{P}, l, i)\right) \tag{5.1}$$

$$v(\mathbf{P}, l, i) = \sum_j^{|\mathbf{T}|} \mathbf{w}[l, i, j] \cdot \mathbf{x}(\mathbf{P} \circ \mathbf{T}[j]) \tag{5.2}$$

$$\mathbf{x}(\mathbf{P}') = \left\langle \begin{array}{c} S(M[l-1, 0])(\mathbf{P}') \\ \dots \\ S(M[l-1, I_{l-1}])(\mathbf{P}') \end{array} \right\rangle \tag{5.3}$$

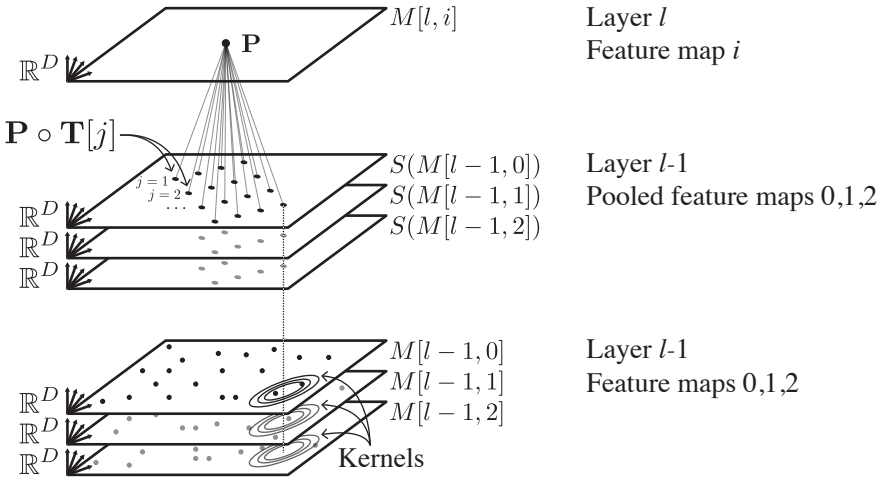

Figure 5.1: The evaluation of point $\mathbf{P}$ in map $M[l,i]$. The elements of the k-neighborhood of $\mathbf{P}$ are computed $\mathbf{P} \circ \mathbf{T}[j]$. Each point in the neighborhood is evaluated in the pooled feature maps of the lower layer $l-1$. The pooled maps are computed with kernels on the underlying feature maps. The dashed line intersects the points in the pooled map whose values form $\mathbf{x}(\mathbf{P} \circ \mathbf{T}[j])$ in Equation 3; it also intersects the contours of kernels used to compute those pooled values. The value of the feature is the sum of the dot-products $\mathbf{w}[l,i,j] \cdot \mathbf{x}(\mathbf{P} \circ \mathbf{T}[j])$ over all j, followed by a nonlinearity.

where σ is the nonlinearity (e.g., $\tanh(x)$ or $\max(x,0)$), $v(\mathbf{P},l,i)$ is the dot product, $\mathbf{P} \circ \mathbf{T}[j]$ represents element j in the k-neighborhood of $\mathbf{P}$, and $\mathbf{x}(\mathbf{P'})$ is the vector of values from the underlying pooled maps at point $\mathbf{P'}$. This definition is a generalization of feature maps in convnets[1]. Similarly, the same filter weights $\mathbf{w}[l,i,j]$ are tied across all points $\mathbf{P}$ in feature map $M[l,i]$. The evaluation of a point in a feature map is visualized in Figure 1.

Feature maps $M[l,i]$ are pooled via kernel convolution to become $S(M[l,i])$. In the case of sum-pooling, $S(M[l,i])(\mathbf{P}) = \int M[l,i](\mathbf{P}-\mathbf{Q})K(\mathbf{Q})\,\mathrm{d}\mathbf{Q}$; for max-pooling, $S(M[l,i])(\mathbf{P}) = \max_Q M[l,i](\mathbf{P}-\mathbf{Q})K(\mathbf{Q})$. The kernel $K(\mathbf{Q})$ is also a scalar function of the D-dimensional symmetry space. In the previous example of a person feature, the arm feature map could

[1]The neighborhood that defines a square filter in convnets is the reference point translated by up to k times in x and k times in y.

be pooled over a wide range of rotations but narrow range of translations and scales so that the person feature allows for moveable but not unrealistic arms. Each filter can specify the kernels it uses to pool lower layers, but for the sake of brevity and analogy to convnets we assume that the feature maps of a layer are pooled by the same kernel. Note that convnets discretize these operations, subsample the pooled map, and use a uniform kernel $K(\mathbf{Q}) = \mathbb{1}\{\|\mathbf{Q}\|_\infty < r\}$.

As with convnets, the values of points in a symnet feature map are used by higher symnet layers, layers of fully connected hidden units, and ultimately softmax classification. Hidden units take the familiar form $\mathbf{o} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, with input $\mathbf{x}$, output $\mathbf{o}$, weight matrix $\mathbf{W}$, and bias $\mathbf{b}$.

The log-loss of the softmax $\mathcal{L}$ on an instance is $-\mathbf{w_i} \cdot \mathbf{x} - b_i + \log\left(\sum_c \exp\left(\mathbf{w_c} \cdot \mathbf{x} + b_c\right)\right)$, where $Y = i$ is the true label, $\mathbf{w_c}$ and b_c are the weight vector and bias for class c, and the summation is over the classes. The input image is treated as a feature map (or maps, if color or stereo) with values in the translation symmetry space.

Deep symmetry networks are trained with backpropagation and are amenable to the same best practices as convnets. Though feature maps are defined as continuous, in practice the maps and their gradients are evaluated on a finite set of points $\mathbf{P} \in M[l,i]$. We provide the partial derivative of the loss $\mathcal{L}$ with respect to a weight vector.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}[l,i,j]} = \sum_{\mathbf{P} \in M[l,i]} \frac{\partial \mathcal{L}}{\partial M[l,i](\mathbf{P})} \frac{\partial M[l,i](\mathbf{P})}{\partial \mathbf{w}[l,i,j]} \tag{5.4}$$

$$\frac{\partial M[l,i](\mathbf{P})}{\partial \mathbf{w}[l,i,j]} = \sigma'\left(v(\mathbf{P},l,i)\right) \mathbf{x}(\mathbf{P} \circ \mathbf{T}[j]) \tag{5.5}$$

The partial derivative of the loss $\mathcal{L}$ with respect to the value of a point in a lower layer is

$$\frac{\partial \mathcal{L}}{\partial M[l-1,i](\mathbf{P})} = \sum_{i'}^{I_l} \sum_{\mathbf{P}' \in M[l,i']} \frac{\partial \mathcal{L}}{\partial M[l,i'](\mathbf{P}')} \frac{\partial M[l,i'](\mathbf{P}')}{\partial M[l-1,i](\mathbf{P})} \tag{5.6}$$

$$\frac{\partial M[l,i'](\mathbf{P}')}{\partial M[l-1,i](\mathbf{P})} = \sigma'\left(v(\mathbf{P}',\mathbf{l},\mathbf{i}')\right) \sum_j^{|\mathbf{T}|} \mathbf{w}[l,i',j][i] \frac{\partial S(M[l-1,i])(\mathbf{P}' \circ \mathbf{T}[j])}{\partial M[l-1,i](\mathbf{P})} \tag{5.7}$$

where the gradient of the pooled feature map $\frac{\partial S(M[l,i])(\mathbf{P})}{\partial M[l,i](\mathbf{Q})}$ equals $K(\mathbf{P} - \mathbf{Q})$ for sum-pooling.

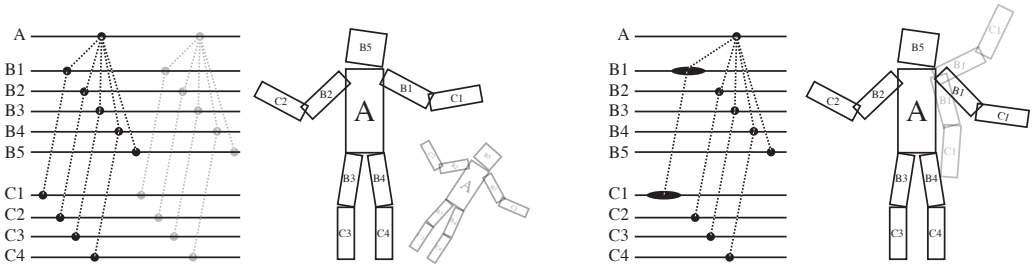

Figure 5.2: The feature hierarchy of a three-layer deep affine net is visualized with and without pooling. From top to bottom, the layers (A,B,C) contain one, five, and four feature maps, each corresponding to a labeled part of the cartoon figure. Each horizontal line represents a six-dimensional affine feature map, and bold circles denote six-dimensional points in the map. The dashed lines represent the affine transformation from a feature to the location of one of its filter points. For clarity, only a subset of filter points are shown. **Left:** Without pooling, the hierarchy represents a rigid affine transformation among all maps. Another point on feature map A is visualized in grey. **Right:** Feature maps B1 and C1 are pooled with a kernel that gives those features flexibility in rotation.

None of this treatment depends explicitly on the dimensionality of the space except for the kernel and transformation composition which have polynomial dependence on D. In the next section we apply this architecture to the affine group in 2D, but it could also be applied to the affine group in 3D or any other symmetry group.

5.4 Deep Affine Networks

We instantiate a deep symmetry network with the affine symmetry group in the plane. The affine symmetry group contains transformations capable of rotating, scaling, shearing, and translating two-dimensional points. The transformation is described by six coordinates:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

This means that each of the feature maps $M[l, i]$ and elements $\mathbf{T}[j]$ of the k-neighborhood is represented in six dimensions. The identity transformation is $a=d=1, b=c=e=f=0$. The

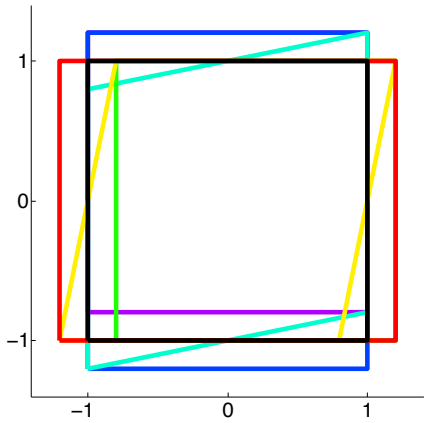

Figure 5.3: The six transformations in the generating set of the affine group applied to a square (exaggerated $\epsilon = 0.2$, identity is black square).

generating set of the affine symmetry group contains six elements, each of which is obtained by adding ϵ to one of the six coordinates in the identity transform. This generating set is visualized in Figure 3.

A deep affine network can represent a rich part hierarchy where each weight of a feature modulates the response to a subpart at a point in the affine neighborhood. The geometry of a deep affine network is best understood by tracing a point on a feature map through its filter point transforms into lower layers. Figure 2 visualizes this structure without and with pooling on the left and right sides of the diagram, respectively. Without pooling, the feature hierarchy defines a rigid affine relationship between the point of evaluation on a map and the location of its sub-features. In contrast, a pooled value on a sub-feature map is computed from a neighborhood defined by the kernel of points in affine space; this can represent model flexibility along certain dimensions of affine space.

5.5 Scaling to High-Dimensional Symmetry Spaces

It would be intractable to explicitly represent the high-dimensional feature maps of symnets. Even a subsampled grid becomes unwieldy at modest dimensions (e.g., a grid in affine space with ten steps per axis has 10^6 points). Instead, each feature map is evaluated at N control

points. The control points are local maxima of the feature in symmetry space, found by Gauss-Newton optimization, each initialized from a prior. This can be seen as a form of non-maximum suppression. Since the goal is recognition, there is no need to approximate the many points in symmetry space where the feature is not present. The map is then interpolated with kernel functions; the shape of the function also controls pooling.

5.5.1 Transformation Optimization

Convnets max-pool a neighborhood of translation space by exhaustive evaluation of feature locations. There are a number of algorithms that solve for a maximal feature location in symmetry space but they are not efficient when the feature weights are frequently adjusted [48, 100]. We adopt an iterative approach that dovetails with the definition of our features.

If a symnet is based on a Lie group, gradient based optimization can be used to find a point $\mathbf{P}^*$ that locally maximizes the feature value (Equation 1) initialized at point $\mathbf{P}$. In our experiments with deep affine nets, we follow the forward compositional (FC) warp [8] to align filters with the image. An extension of Lucas-Kanade, FC solves for an image alignment. We adapt this procedure to our filters and weight vectors: $\min_{\Delta\mathbf{P}} \sum_j^{|\mathbf{T}|} \|\mathbf{w}[l,i,j] - \mathbf{x}(\mathbf{P} \circ \Delta\mathbf{P} \circ \mathbf{T}[j])\|^2$. We run an FC alignment for each of the N control points in feature map $M[l,i]$, each initialized from a prior. Assuming $\sum_j^{|\mathbf{T}|} \|\mathbf{x}(\mathbf{P} \circ \Delta\mathbf{P} \circ \mathbf{T}[j])\|^2$ is constant, this procedure locally maximizes the dot product between the filter and the map in Equation 2. Each iteration of FC takes a Gauss-Newton step to solve for a transformation of the neighborhood of the feature in the underlying map $\Delta\mathbf{P}$, which is then composed with the control point: $\mathbf{P} \leftarrow \mathbf{P} \circ \Delta\mathbf{P}$.

5.5.2 Kernels

Given a set of N local optima $O^* = \{(\mathbf{P}_1, v_1), \ldots, (\mathbf{P}_N, v_N)\}$ in D-dimensional feature map $M[l,i]$, we use kernel-based interpolation to compute a pooled map $S(M[l,i])$. The kernel performs three functions: penalizing relative locations of sub-features in symmetry space (*cf.* [50]), interpolating the map, and pooling a region of the map. These roles could be split into separate filter-specific kernels that are then convolved appropriately. The choice

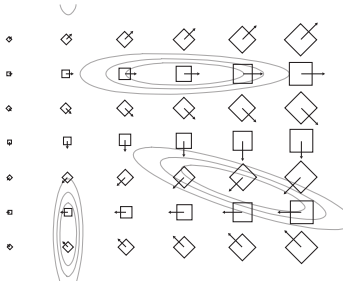

Figure 5.4: Contours of three 6D Gaussian kernels visualized on a surface in affine space. Points are visualized by an oriented square transformed by the affine transformation at that point. Each kernel has a different covariance matrix Σ.

of these kernels will vary with the application. In our experiments, we lump these functions into a single kernel for a layer. We use a Gaussian kernel $K(\mathbf{Q}) = e^{-\mathbf{q}^T\Sigma^{-1}\mathbf{q}}$ where $\mathbf{q}$ is the D-dimensional vector representation of $\mathbf{Q}$ and the $D \times D$ covariance matrix Σ controls the shape and extent of the kernel. Several instances of this kernel are shown in Figure 4. Max-pooling produced the best results on our tests.

5.6 Related Work

We share with other researchers the hypothesis that explanatory factors cannot be disentangled unless they are represented in an appropriate symmetry space [12, 69]. Our adaptation of a representation to work in symmetry space is similar in some respects to the use of tangent distance in nearest-neighbor classifiers [138]. Symnets, however, are deep networks that compute features in symmetry space at every level. Whereas the tangent distance approximation is only locally accurate, symnet feature maps can represent large displacements in symmetry space. There are other deep networks that reinterpret the invariance of convolutional networks. Scattering networks [21] are cascades of wavelet decompositions designed to be invariant to particular Lie groups, where translation and rotation invariance have been demonstrated so far. The M-theory of Anselmi et al. [5] constructs features invariant to a symmetry group by using statistics of dot products with group orbits. We differ from these networks in that we model multiple symmetries jointly in each layer, we do not completely

pool out a symmetry, and we discriminatively train our entire architecture. The first two differences are important because objects and their subparts may have relative flexibility but not total invariance along certain dimensions of symmetry space. For example, a leg of a person can be seen in some but not all combinations of rotation and scale relative to the torso. Without discriminative training, scattering networks and M-theory are limited to representing features whose invariances may be inappropriate for a target concept because they are fixed ahead of time, either by the wavelet hierarchy of the former or unsupervised training of the latter. The discriminative training of symnets yields features with task-oriented invariance to their sub-features. In the context of digit recognition this might mean learning the concept of a '0' with more rotation invariance than a '6', which would incur loss if it had positive weights in the region of symmetry space where a '9' would also fire.

Much of the vision literature is devoted to features that reduce or remove the effects of certain symmetry groups, e.g., [99, 95]. Each feature by itself is not discriminative for object recognition, so structure is modeled separately, usually with a representation that does not generalize to novel viewpoints (e.g., bags-of-features) or with a rigid alignment algorithm that cannot represent uncertainty over geometry (e.g. [48, 100]). Compared to symnets, these features are not learned, have invariance limited to a small set of symmetries, and destroy information that could be used to model object sub-structure. Like deformable part models [50], symnets can model and penalize relative transformations that compose up the hierarchy, but can also capture additional symmetries.

Symmetry group theory has made a limited number of appearances in machine learning [46]. A few applications are discussed by Kondor [81], and they are also used in determinantal point processes [86]. Methods for learning transformations from examples [165, 69] could potentially benefit from being embedded in a deep symmetry network. Symmetries in graphical models [110] lead to effective lifted probabilistic inference algorithms. Deep symmetry networks may be applicable to these and other areas.

5.7 Experiments

In our experiments we test the hypothesis that a deep network with access to a larger symmetry group will generalize better from fewer examples, provided those symmetries are present in the data. In particular, theory suggests that a symnet will have better sample complexity than another classifier on a dataset if it is based on a symmetry group that generates variations present in that dataset [2]. We compare deep affine symnets to convnets on the MNIST-rot and NORB image classification datasets, which finely sample their respective symmetry spaces such that learning curves measure the amount of augmentation that would be required to achieve similar performance. On both datasets affine symnets achieve a substantial reduction in sample complexity. This is particularly remarkable on NORB because its images are generated by a symmetry space in 3D. Symnet running time was within an order of magnitude of convnets, and could be greatly optimized.

5.7.1 MNIST-rot

MNIST-rot [88] consists of 28x28 pixel greyscale images: 10^4 for training, 2×10^3 for validation, and 5×10^4 for testing. The images are sampled from the MNIST digit recognition dataset and each is rotated by a random angle in the uniform distribution $[0, 2\pi]$. With transformations that apply to the whole image, MNIST-rot is a good testbed for comparing the performance of a single affine layer to a single convnet layer.

We modified the Theano [13] implementation of convolutional networks so that the network consisted of a single layer of convolution and maxpooling followed by a hidden layer of 500 units and then softmax classification. The affine net layer was directly substituted for the convolutional layer. The control points of the affine net were initialized at uniformly random positions with rotations oriented around the image center, and each control point was locally optimized with four iterations of Gauss-Newton updates. The filter points of the affine net were arranged in a square grid. Both the affine net and the convnet compute a dot-product and use the sigmoid nonlinearity. Both networks were trained with 50 epochs of

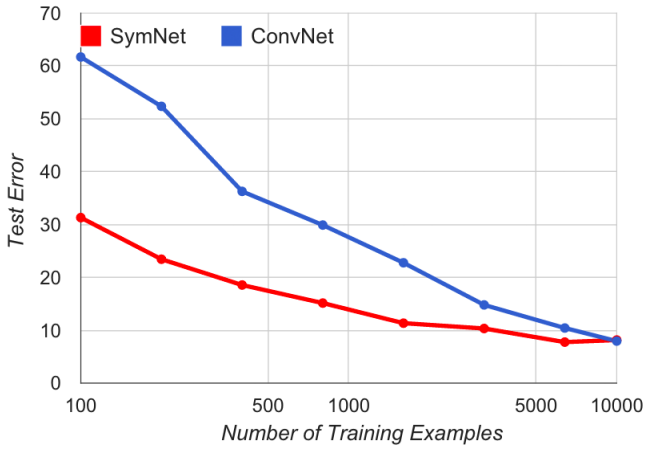

Figure 5.5: Impact of training set size on MNIST-rot test performance for architectures that use either one convolutional layer or one affine symnet layer.

mini-batch gradient descent with momentum, and test results are reported on the network with lowest error on the validation set[2]. The convnet did best with small 5×5 filters and the symnet with large 20×20 filters. This is not surprising because the convnet must approximate the large rotations of the dataset with translations of small patches. The affine net can pool directly in this space of rotations with large filters.

Learning curves for the two networks are presented in Figure 5. We observe that the affine symnet roughly halves the error of the convnet. With small sample sizes, the symnet achieves an accuracy for which the convnet requires about eight times as many samples.

5.7.2 NORB

MNIST-rot is a synthetic dataset with symmetries that are not necessarily representative of real images. The NORB dataset [92] contains $2 \times 108 \times 108$ pixel stereoscopic images of 50 toys in five categories: quadrupeds, human figures, airplanes, trucks, and cars. Five of the ten instances of each category are reserved for the test set. Each toy is photographed

[2]Grid search over learning rate $\{.1, .2\}$, mini-batch size $\{10, 50, 100\}$, filter size $\{5, 10, 15, 20, 25\}$, number of filters $\{20, 50, 80\}$, pooling size (convnet) $\{2, 3, 4\}$, and number of control points (symnet) $\{5, 10, 20\}$.

on a turntable from an exhaustive set of angles and lighting conditions. Each image is then perturbed by a random translation shift, planar rotation, luminance change, contrast change, scaling, distractor object, and natural image background. A sixth blank category containing just the distractor and background is also used. As in other papers, we downsample the images to $2 \times 48 \times 48$. To compensate for the effect of distractors in smaller training sets, we also train and test on a version of the dataset that is centrally-cropped to $2 \times 24 \times 24$. We report results for whichever version had lower validation error. In our experiments we train on a variable subset of the first training fold, using the first 2×10^3 images of the second fold for validation. Our results use both of the testing folds.

We compare architectures that use two convolutional layers or two affine ones, which performed better than single-layer ones. As with the MNIST-rot experiments, the symnet and convnet layers are followed by a layer of 500 hidden units and softmax classification. The symnet control points in the first layer were arranged in three concentric rings in translation space, with 8 points spaced across rotation (200 total points). Control points in the second layer were fixed at the center of translation space arranged over 8 rotations and up to 2 vertical scalings (16 total points) to approximate the effects of elevation change. Control points were not iteratively optimized due to the small size of object parts in downsampled images. The filter points of the first layer of the affine net were arranged in a square grid. The second layer filter points were arranged in a circle in translation space at a 3 or 4 pixel radius, with 8 filter points evenly spaced across rotation at each translation. We report the test results of the networks with lowest validation error on a range of hyperparameters[3].

The learning curves for convnets and affine symnets are shown in Figure 6. Even though the primary variability in NORB is due to rigid 3D transformations, we find that our affine networks still have an advantage over convnets. A 3D rotation can be locally approximated with 2D scales, shears, and rotations. The affine net can represent these transformations and so it benefited from larger filter patches. The translation approximation of the convnet

[3]Grid search over filter size in each layer $\{6, 9\}$, pooling size in each layer (convnet) $\{2, 3, 4\}$, first layer control point translation spacing (symnet) $\{2, 3\}$, momentum $\{0, 0.5, 0.9\}$, others as in MNIST-rot.

is unable to properly align larger features to the true symmetries, and so it performed better with smaller filters. The convnet requires about four times as much data to reach the accuracy of the symnet with the smallest training set. Larger filters capture more structure than smaller ones, allowing symnets to generalize better than convnets, and effectively giving each symnet layer the power of more than one convnet layer.

The left side of the graph may be more indicative of the types of gains symnets may have over convnets in more realistic datasets that do not have thousands of images of identical 3D shapes. With the ability to apply more realistic transformations to sub-parts, symnets may also be better able to reuse substructure on datasets with many interrelated or fine-grained categories. Since symnets are a clean generalization of convnets, they should benefit from the learning, regularization, and efficiency techniques used by state-of-the-art networks [84].

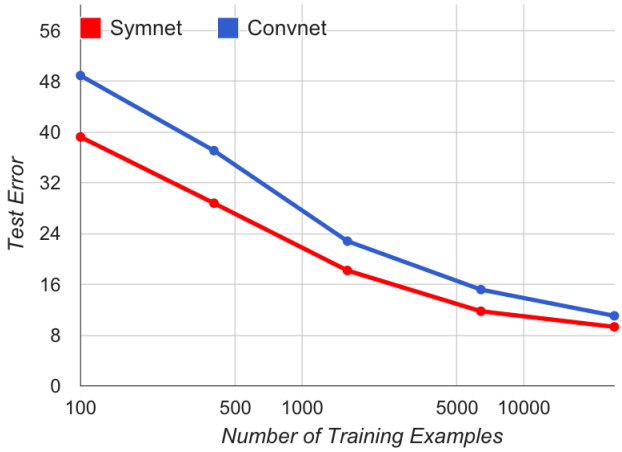

Figure 5.6: Impact of training set size on NORB test performance for architectures with two convolutional or affine symnet layers followed by a fully connected layer and then softmax classification.

5.8 Conclusion

Symmetry groups underlie the hardest challenges in computer vision. In this chapter we introduced deep symmetry networks, the first deep architecture that can compute features

over any symmetry group. It is a natural generalization of convolutional neural networks that uses kernel interpolation and transformation optimization to address the difficulties in representing high-dimensional feature maps. In experiments on two image datasets with 2D and 3D variability, affine symnets achieved higher accuracy than convnets while using significantly less data.

Chapter 6

COMPOSITIONAL KERNEL MACHINES

6.1 Introduction

Despite impressive results on classification benchmarks, state of the art convnets have drawbacks. The depth of these networks is a double-edged sword: it yields both nonlinearity for sophisticated discrimination and nonconvexity for frustrating optimization. The established training procedure for ILSVRC classification cycles through the million-image training set more than fifty times, requiring substantial stochasticity, data augmentation, and hand-tuned learning rates. On today's consumer hardware, the process takes several days. However, performance depends heavily on hyperparameters, which include the number and connections of neurons as well as optimization details. Unfortunately, the space of hyperparameters is unbounded, and each configuration of hyperparameters requires the aforementioned training procedure. It is no surprise that large organizations with enough computational power to conduct this search dominate this task.

Yet mastery of object recognition on a static dataset is not enough to propel robotics and internet-scale applications with ever-growing instances and categories. Each time the training set is modified, the convnet must be retrained ("fine-tuned") for optimum performance. If the training set grows linearly with time, the total training computation grows quadratically. Convnets can be easily tricked into producing spurious high confidence classifications. One method adds imperceptible perturbations to an image to change the convnet's response [149]. Another generates nonsense tilings and noise patterns [109].

We propose the Compositional Kernel Machine (CKM), a kernel-based visual classifier that has the symmetry and compositionality of convnets and symnets but with the training benefits of instance-based learning (IBL). CKMs branch from the original instance-based

methods with *virtual instances*, an exponential set of plausible compositions of training instances. The first steps in this direction are promising compared to IBL and deep methods, and future work will benefit from over fifty years of research into nearest neighbor algorithms, kernel methods, and neural networks. Just as support vector machines (SVMs) eclipsed multilayer perceptrons in the 1990s, CKMs could become a compelling alternative to convnets with reduced training time and sample complexity.

In this chapter we first define CKMs, explore their formal and computational properties, and compare them to existing kernel methods. We then propose a key contribution of this work: a sum-product function (SPF) that efficiently sums over an exponential number of virtual instances. We then describe how to train the CKM with and without parameter optimization. Finally, we present results on NORB and its variations that show CKMs can be competitive with deep models and can outperform them on tests of composition and symmetry.

6.2 Compositional Kernel Machines

The key issue in using an instance-based learner on large images is the curse of dimensionality. Even the millions of training images of ImageNet are not enough to construct a meaningful neighborhood for a downscaled 256×256 image. The compositional kernel machine (CKM) addresses this issue by constructing an exponential number of *virtual instances*. The core hypothesis is that the variation of the visual world can be understood as a rearrangement of low-dimensional pieces that have been seen before. For example, an image of a car could be recognized by matching many pieces from other images of cars from different viewpoints. The virtual instances represent this set of all possible transformations and recombinations of the training images. The arrangement of these pieces cannot be arbitrary, so CKMs learn how to compose virtual instances with weights on compositions. A major contribution of this work is the ability to efficiently sum over this set with a sum-product function.

The set of virtual instances is related to the nonlinear image manifolds described by Simard et al. [138] but with key differences. Whereas the tangent distance accounts for

transformations applied to the whole image, virtual instances can depict local transformations that are applied differently across an image. Secondly, the tangent distance is a linearization of the nonlinear transformation which is only accurate near the training images. Virtual instances can easily represent non-local transformations. Unlike the explicit augmentation of virtual support vectors in Schölkopf et al. [135], the set of virtual instances in a CKM is implicit and exponentially larger. Platt and Allen [118] demonstrated an early version of virtual instances to expand the set of negative examples for a linear classifier.

6.2.1 Definition

We define CKMs using notation common to other IBL techniques. The two prototypical instance-based learners are k-nearest neighbors and support vector machines. The foundation for both algorithms is a similarity or kernel function $K(x, x')$ between two instances. Given a training set of m labeled instances of the form $\langle x_i, y_i \rangle$ and query x_q, the k-NN algorithm outputs the most common label of the k nearest instances:

$$y_{\text{kNN}}(x_q) = \arg\max_c \sum_{i=1}^{m} \mathbb{1}\left[c = y_i \wedge K(x_i, x_q) \geq K(x^k, x_q)\right]$$

where $\mathbb{1}[\cdot]$ equals one if its argument is true and zero otherwise, and x^k is the k^{th} nearest training instance to query x_q assuming unique distances. The multiclass support vector machine in its dual form can be seen as a weighted nearest neighbor that outputs the class with the highest weighted sum of kernel values with the query:

$$y_{\text{SVM}}(x_q) = \arg\max_c \sum_{i=1}^{m} \alpha_{i,c} K(x_i, x_q) \tag{6.1}$$

where $\alpha_{i,c}$ is the weight on training instance x_i that contributes to the score of class c [36].

The CKM performs the same classification as these instance-based methods but it sums over an exponentially larger set of virtual instances to mitigate the curse of dimensionality. Virtual instances are composed of rearranged elements from one or more training instances.

Depending on the design of the CKM, elements can be subsets of instance variables (e.g., overlapping pixel patches) or features thereof (e.g., ORB features or a 2D grid of convnet feature vectors). We assume there is a deterministic procedure that processes each training or test instance x_i into a fixed tuple of indexed elements $E_{x_i} = (e_{i,1},\ \ldots,\ e_{i,|E_{x_i}|})$, where instances may have different numbers of elements. The query instance x_q (and its corresponding tuple of elements E_{x_q}) is the example that is being classified by the CKM; it is a training instance during training and a test instance during testing. A virtual instance z is represented by a tuple of elements from training instances, e.g. $E_z = (e_{10,5},\ e_{71,2},\ \ldots,\ e_{46,17})$. Given a query instance x_q, the CKM represents a set of virtual instances each with the same number of elements as E_{x_q}. We define a leaf kernel $K_L(e_{i,j}, e_{i',j'})$ that measures the similarity between any two elements. Using kernel composition [6], we define the kernel between the query instance x_q and a virtual instance z as the product of the leaf kernels over their corresponding elements: $K(z, x_q) = \prod_j^{|E_{x_q}|} K_L(E_z[j], E_{x_q}[j])$, where $E[j]$ denotes the j$^{\text{th}}$ element.

We combine leaf kernels with weighted sums and products to compactly represent a sum over kernels with an exponential number of virtual instances. Just as a Sum-Product Network can compactly represent a mixture model that is a weighted sum over an exponential number of mixture components, the same algebraic decomposition can compactly encode a weighted sum over an exponential number of kernels. For example, if the query instance is represented by two elements $E_{x_q} = (e_{q,1},\ e_{q,2})$ and the training set contains elements $\{e_1,\ e_2,\ e_3,\ e_4,\ e_5,\ e_6\}$, then

$$\begin{aligned}&[w_1 K_L(e_{q,1}, e_1) + w_2 K_L(e_{q,1}, e_2) + w_3 K_L(e_{q,1}, e_3)] \times \\ &\quad [w_4 K_L(e_{q,2}, e_4) + w_5 K_L(e_{q,2}, e_5) + w_6 K_L(e_{q,2}, e_6)]\end{aligned}$$

expresses a weighted sum over nine virtual instances using eleven additions/multiplications instead of twenty-six for an expanded flat sum $w_1 K_L(e_{q,1}, e_1) K_L(e_{q,2}, e_4) + \ldots + w_9 K_L(e_{q,1}, e_3)$ $K_L(e_{q,2}, e_6)$. If the query instance and training set contained 100 and 10000 elements, respectively, then a similar factorization would use $O(10^6)$ operations compared to a naïve sum

over the 10^{500} virtual instances. Leveraging the Sum-Product Theorem [54], we define CKMs to allow for more expressive architectures with this exponential computational savings.

CKMs are defined recursively: (1) a leaf kernel over a query element and a training set element is a CKM, (2) the product of CKMs with disjoint scopes is a CKM, (3) the weighted sum of CKMs with the same scope is a CKM. The scope of an operator is the set of query elements it takes as inputs; it is analogous to the receptive field of a unit in a neural network, but with CKMs the query elements are not restricted to being pixels on the image grid (e.g., they may be defined as a set of extracted ORB features). A leaf kernel has singleton scope, internal nodes have scope over some subset of the query elements, and the root node of the CKM has full scope of all query elements E_{x_q}. This definition allows for rich CKM architectures with many layers to represent elaborate compositions. The value of each sum node child is multiplied by a weight $w_{k,c}$ and optionally a constant cost function $\phi(e_{i,j}, e_{i',j'})$ that rewards certain compositions of elements. Analogous to a multiclass SVM, the CKM has a separate set of weights for each class c in the dataset. The CKM classifies a query instance as $y_{\text{CKM}}(x_q) = \arg\max_c S_c(x_q)$, where $S_c(x_q)$ is the value of the root node of the CKM evaluating query instance x_q using weights for class c.

Definition 5 (Friesen and Domingos [54])**.** *A product node is decomposable iff the scopes of its children are disjoint. An SPF is decomposable iff all of its product nodes are decomposable.*

Theorem 3 (Sum-Product Theorem, Friesen and Domingos [54])**.** *Every decomposable SPF can be summed over its domain in time linear in its size.*

Corollary 1. *$S_c(x_q)$ can sum over the set of virtual instances in time linear in the size of the SPF.*

Proof. For each query instance element $e_{q,j}$ we define a discrete variable Z_j with a state for each training element $e_{i',j'}$ found in a leaf kernel $K_L(e_{q,j}, e_{i',j'})$ in the CKM. The Cartesian product of the domains of the variables $\boldsymbol{Z}$ defines the set of virtual instances represented by the CKM. $S_c(x_q)$ is a SPF over semiring $(R, \oplus, \otimes, 0, 1)$, variables $\boldsymbol{Z}$, constant functions $\boldsymbol{w}$ and

$\boldsymbol{\phi}$, and univariate functions $K_L(e_{q,j}, Z_j)$. With the appropriate definition of leaf kernels, any semiring can be used. The definition above provides that the children of every product node have disjoint scopes. Constant functions have empty scope so there is no intersection with scopes of other children. With all product nodes decomposable, $S_c(x_q)$ is a decomposable SPF and can therefore sum over all states of $\boldsymbol{Z}$, the virtual instances, in time linear to the size of the CKM. □

Special cases of CKMs include multiclass SVMs (flat sum-of-products) and local naive Bayes [102] (flat product-of-sums). A CKM can be seen as a generalization of an image grammar [55] where terminal symbols corresponding to pieces of training images are scored with kernels and non-terminal symbols are sum nodes with a production for each child product node.

The weights and cost functions of the CKM control the weights on the virtual instances. Each virtual instance represented by the CKM defines a tree that connects the root to the leaf kernels over its unique composition of training set elements. If we were to expand the CKM into a flat sum (cf. Equation 6.1), the weight on a virtual instance would be the product of the weights and cost functions along the branches of its corresponding tree. These weights are important as they can prevent implausible virtual instances. For example, if we use image patches as the elements and allow all compositions, the set of virtual instances would largely contain nonsense noise patterns. If the elements were pixels, the virtual instances could even contain arbitrary images from classes not present in the training set. There are many aspects of composition that can be encoded by the CKM. For example, we can penalize virtual instances that compose training set elements using different symmetry group transformations. We could also penalize compositions that juxtapose elements that disagree on the contents of their borders. Weights can be learned to establish clusters of elements and reward certain arrangements. In Section 6.3 we demonstrate one choice of weights and cost functions in a CKM architecture built from extracted image features.

6.2.2 Learning

The training procedure for a CKM builds an SPF that encodes the virtual instances. There are then two options for how to set weights in the model. As with k-NN, the weights in the CKM could be set to uniform. Alternatively, as with SVMs, the weights could be optimized to improve generalization and reduce model size.

For weight learning, we use block-coordinate gradient descent to optimize leave-one-out loss over the training set. The leave-one-out loss on a training instance x_i is the loss on that instance made by the learner trained on all data except x_i. Though it is an almost unbiased estimate of generalization error [101], it is typically too expensive to compute or optimize with non-IBL methods [23]. With CKMs, caching the SPFs and efficient data structures make it feasible to compute exact partial derivatives of the leave-one-out loss over the whole training set. We use a multiclass squared-hinge loss

$$\mathcal{L}(x_i, y_i) = \max\left[1 + \underbrace{S_{y'}(x_i)}_{\text{Best incorrect class}} - \underbrace{S_{y_i}(x_i)}_{\text{True class}}, 0\right]^2$$

for the loss on training instance x_i with true label y_i and highest-scoring incorrect class y'. We use the squared version of the hinge loss as it performs better empirically and prioritizes updates to element weights that led to larger margin violations. In general, this objective is not convex as it involves the difference of the two discriminant functions which are strictly convex (due to the choice of semiring and the product of weights on each virtual instance). In the special case of the sum-product semiring and unique weights on virtual instances the objective is convex as is true for L2-SVMs. Convnets have a non-convex objective, but they require lengthy optimization to perform well. As we show in Section 6.3, CKMs can achieve high accuracy with uniform weights, which further serves as good initialization for gradient descent.

For each epoch, we iterate through the training set, for each training instance x_i optimizing the block of weights on those branches with E_{x_i} as descendants. We take gradient

steps to lower the leave-one-out loss over the rest of the training set $\sum_{i' \in ([1,m] \setminus i)} \mathcal{L}(x_{i'}, y_{i'})$. We iterate until convergence or an early stopping condition. A component of the gradient of the squared-hinge loss on an instance takes the form

$$\frac{\partial}{\partial w_{k,c}} \mathcal{L}(x_i, y_i) = \begin{cases} 2\Delta(x_i, y_i) \frac{\partial S_{y'}(x_i)}{\partial w_{k,c}} & \Delta(x_i, y_i) > 0 \wedge c = y' \\ -2\Delta(x_i, y_i) \frac{\partial S_{y_i}(x_i)}{\partial w_{k,c}} & \Delta(x_i, y_i) > 0 \wedge c = y_i \\ 0 & \text{otherwise} \end{cases}$$

where $\Delta(x_i, y_i) = 1 + S_{y'}(x_i) - S_{y_i}(x_i)$. As developed in Chapter 3, we compute partial derivatives $\frac{\partial S_c(x_i)}{\partial w_{k,c}}$ with backpropagation through the SPF. For efficiency, terms of the gradient can be set to zero and excluded from backpropagation if the values of corresponding leaf kernels are small enough. This is either exact (e.g., if $\oplus$ is maximization) or an approximation (e.g., if $\oplus$ is normal addition).

6.2.3 Scalability

CKMs have several scalability advantages over convnets. As mentioned previously, they do not require a lengthy training procedure. This makes it much easier to add new instances and categories. Whereas most of the computation to evaluate a single setting of convnet hyperparameters is sunk in training, CKMs can efficiently race hyperparameters on hold-out data [94].

The evaluation of the CKM depends on the structure of the SPF, the size of the training set, and the computer architecture. A basic building block of these SPFs is a sum node with a number of children on the order of magnitude of the training set elements $|\mathcal{E}|$. On a sufficiently parallel computer, assuming the size of the training set elements greatly exceeds the dimensionality of the leaf kernel, this sum node will require $O(\log(|\mathcal{E}|))$ time (the depth of a parallel $\oplus$ reduction circuit) and $O(|\mathcal{E}|)$ space. Duda et al. [49] describe a constant time nearest neighbor circuit that relies on precomputed Voronoi partitions, but this has impractical space requirements in high dimensions. As with support vector machines, optimization

Table 6.1: Dataset Properties

Name	**#Train-#Test**	**Dimensions**	**Classes**
Small NORB	24300-24300	96×96	5
NORB Compositions	100-1000	256×256	2
NORB Symmetries	$\{50, 100, \ldots, 12800\}$-2916	108×108	6

of sparse element weights can greatly reduce model size.

On a modest multicore computer, we must resort to using specialized data structures. Hash codes can be used to index raw features or to measure hamming distance as a proxy to more expensive distance functions. While they are perhaps the fastest method to accelerate a nearest neighbor search, the most accurate hashing methods involve a training period yet do not necessarily result in high recall [156, 66]. There are many space-partitioning data structure trees in the literature, however in practice none are able to offer exact search of nearest neighbors in high dimensions in logarithmic time. A good compromise of speed and accuracy can be found with approximate search of multiple hierarchical k-means trees [106], which we use in our experiments.

6.3 Experiments

We test CKMs on three image classification scenarios that feature images from either the "small" NORB dataset or the "jittered cluttered" NORB dataset [92]. Both NORB datasets contain greyscale images of five categories of plastic toys photographed with varied altitudes, azimuths, and lighting conditions. Table 6.1 summarizes the datasets. We first describe the SPN architecture and then detail each of the three scenarios.

6.3.1 Experimental Architecture

In our experiments the architecture of the SPF $S_c(x_q)$ for each query image is based on its unique set of extracted ORB features. Like SIFT features, ORB features are rotation-invariant and produce a descriptor from intensity differences, but ORB is much faster to com-

pute and thus suitable for realtime applications [129]. The elements $E_{x_i} = (e_{i,1}, \ldots, e_{i,|E_i|})$ of each image x_i are its extracted keypoints, where an element's feature vector and image position are denoted $\vec{f}(e_{i,j})$ and $\vec{p}(e_{i,j})$ respectively. We use the max-sum semiring ($\oplus = \max$, $\otimes = +$) because it is more robust to noisy virtual instances, yields sparser gradients, is more efficient to compute, and performs better empirically compared with the sum-product semiring.

The SPF $S_c(x_q)$ maximizes over variables $\boldsymbol{Z} = (Z_1, \ldots, Z_{|E_{x_q}|})$ corresponding to query elements E_{x_q} with states corresponding to all possible virtual instances. The SPF contains a unary scope max node for every variable $\{Z_j\}$ that maximizes over the weighted kernels of all possible training elements $\mathcal{E}$: $\oplus(Z_j) = \bigoplus_{z_j \in \mathcal{E}} w_{z_j,c} \otimes K_L(z_j, e_{q,j})$. The SPF contains a binary scope max node for all pairs of variables $\{Z_j, Z_{j'}\}$ for which at least one corresponding query element is within the k-nearest spatial neighbors of the other. These nodes maximize over the weighted kernels of all possible combinations of training set elements.

$$\oplus(Z_j, Z_{j'}) = \bigoplus_{z_j \in \mathcal{E}} \bigoplus_{z_{j'} \in \mathcal{E}} w_{z_j,c} \otimes w_{z_{j'},c} \otimes \phi(z_j, z_{j'}) \otimes K_L(z_j, e_{q,j}) \otimes K_L(z_{j'}, e_{q,j'}) \tag{6.2}$$

This maximizes over all possible pairs of training set elements, weighting the two leaf kernels by two corresponding element weights and a cost function. We use a leaf kernel for image elements that incorporates both the Hamming distance between their features and the Euclidean distance between their image positions: $K_L(e_{i,j}, e_{i',j'}) = \max(\beta_0 - \beta_1 d_{\text{Ham}}(\vec{f}(e_{i,j}), \vec{f}(e_{i',j'})), 0) + \max(\beta_2 ||(\vec{p}(e_{i,j}), \vec{p}(e_{i',j'})||, \beta_3)$. This rewards training set elements that look like a query instance element and appear in a similar location, with thresholds for efficiency. This can represent, for example, the photographic bias to center foreground objects or a discriminative cue from seeing sky at the top of the image. We use the pairwise cost function $\phi(e_{i,j}, e_{i',j'}) = \mathbb{1}[i = i']\beta_4$ that rewards combinations of elements that come from the same source training image. This captures the intuition that compositions involving many source images are less coherent and more likely to contain nonsense than those using fewer. The image is represented as a sum of these unary and binary max nodes. The scopes

of children of the sum are restricted to be disjoint, so the children $\{\oplus(Z_1, Z_2), \oplus(Z_2, Z_3)\}$ would be disallowed, for example. This restriction is what allows the SPF to be tractable, and with multiple sums the SPF has high-treewidth. By comparison, a Markov random field expressing these dependencies would be intractable. The root max node of the SPF has P sums as children, each of which has its random set of unary and binary scope max node children that cover full scope $\boldsymbol{Z}$. We illustrate a simplified version of the SPF architecture in Figure 6.1. Though this SPF represents limited image structure, the definition of CKMs allows for much richer architectures as evidenced by previous chapters and the SPN/SPF literature.

In the following sections, we refer to two variants **CKM** and $\mathbf{CKM}_W$. The **CKM** version uses uniform weights $w_{k,c}$, similar to the basic k-nearest neighbor algorithm. The $\mathbf{CKM}_W$ method optimizes weights $w_{k,c}$ as described in section 6.2.2. Both versions restrict weights for class c to be $-\infty$ ($\oplus$ identity) for those training elements not in class c. This constraint ensures that method **CKM** is discriminative (as is true with k-NN) and reduces the number of parameters optimized by $\mathbf{CKM}_W$. The hyperparameters of ORB feature extraction, leaf kernels, cost function, and optimization were chosen using grid search on a validation set.

6.3.2 Small NORB

We use the original train-test separation which measures generalization to new instances of a category (i.e. tested on toy truck that is different from the toys it was trained on). We show promising results in Table 6.2 comparing CKMs to deep and IBL methods. With improvement over k-NN and SVM, the **CKM** and $\mathbf{CKM}_W$ results show the benefit of using virtual instances to combat the curse of dimensionality. We note that the **CKM** variant that does not optimize weights performs nearly as well as the $\mathbf{CKM}_W$ version that does. Since the test set uses a different set of toys, the use of untrained ORB features hurts the performance of the CKM. Convnets have an advantage here because they discriminatively train their lowest level of features and represent richer image structure in their architecture. To become competitive, future work should improve upon this preliminary CKM architecture.

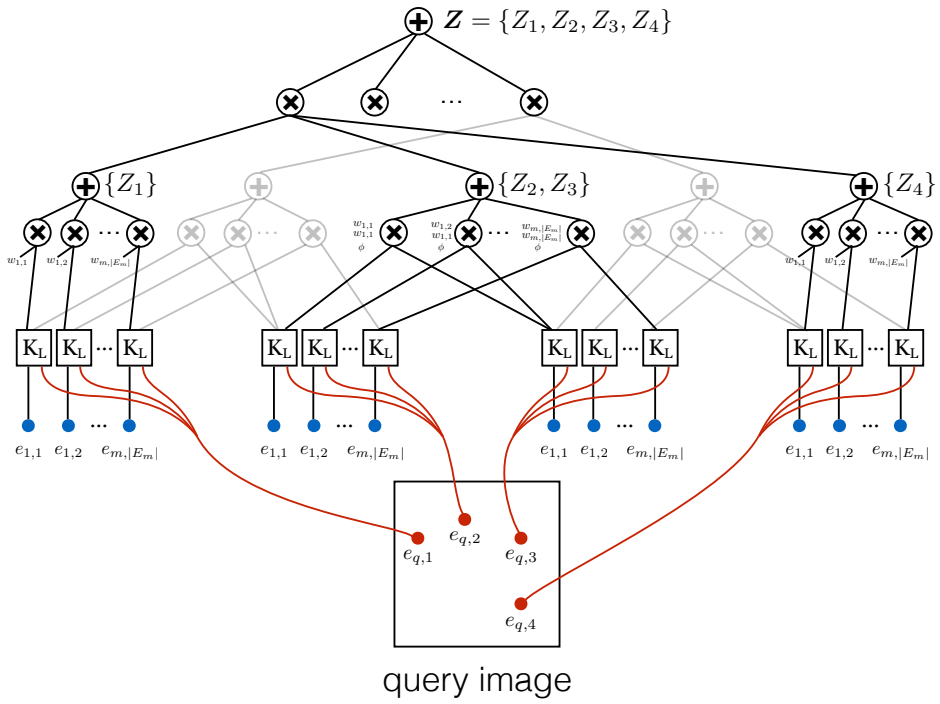

Figure 6.1: Simplified illustration of the SPF $S_c(x_q)$ architecture used in experiments (using ORB features as elements, $|E_{x_q}| \approx 100$). Red dots depict elements E_{x_q} of query instance x_q. Blue dots show training set elements $e_{i,j} \in \mathcal{E}$, duplicated with each query element for clarity. A boxed K_L shows the leaf kernel with lines descending to its two element arguments. The sum nodes are labeled with their scopes. Weights and cost functions (arguments omitted) appear next to product nodes. Only a subset of the unary and binary scope sum nodes are drawn. Only two of the P top-level product nodes are fully detailed (the children of the second are drawn faded).

We demonstrate the advantage of CKMs for representing composition and symmetry in the following experiments.

6.3.3 NORB Compositions

A general goal of representation learning is to disentangle the factors of variation of a signal without having to see those factors in all combinations. To evaluate progress towards this, we created images containing three toys each, sourced from the small NORB training set. Small

Table 6.2: Accuracy on small NORB.

Method	**Accuracy**
Convnet (14 epochs) [10]	94.0%
DBM with aug. training [131]	92.8%
CKM$_W$	89.8%
Convnet (2 epochs) [10]	89.6%
DBM [131]	89.2%
SVM [10]	88.4%
CKM	88.3%
k-NN [92]	81.6%
Logistic regression [92]	77.5%

NORB contains ten types of each toy category (e.g., ten different airplanes), which we divided into two collections. Each image is generated by choosing one of the collections and for each of three categories (person, airplane, animal) randomly sampling a toy from that collection with higher probability than from the other collection (i.e., there are two children with disjoint toy collections but they sometimes borrow). The task is to determine which of the two collections generated the image. This dataset measures whether a method can distinguish different compositions without having seen all possible permutations of those objects through symmetries and noisy intra-class variation. Analogous tasks include identifying people by their clothing, recognizing social groups by their members, and classifying cuisines by their ingredients. We show that CKMs are much more accurate than convnets and k-NN at distinguishing this type of compositional concept.

We compare CKMs to other methods in Table 6.3. Convnets and their features are computed using the TensorFlow library [1]. Training convnets from few images is very difficult without resorting to other datasets; we augment the training set with random crops, which still yields test accuracy near chance. In such situations it is common to train an SVM with features extracted by a convnet trained on a different, larger dataset. We use 2048-dimensional features extracted from the penultimate layer of the pre-trained Inception network [150] and a linear kernel SVM with squared-hinge loss [113]. Notably, the CKM is

much more accurate than the deep methods, and it is about as fast as the SVM despite not taking advantage of the GPU.

Table 6.3: Accuracy on NORB Compositions.

Method	**Accuracy**	**Train+Test (min)**
CKM	82.4%	1.5 [CPU]
SVM with convnet features	75.0%	1 [GPU+CPU]
Convnet	50.6%	9 [GPU]
k-NN on image pixels	51.2%	0.2 [CPU]

Figure 6.2: Images from NORB Compositions

6.3.4 NORB Symmetries

Composition is a useful tool for modeling the symmetries of objects. When we see an image of an object in a new pose, parts of the image may look similar to parts of images of the object in poses we have seen before. In this experiment, we partition the training set of the jittered cluttered NORB into a new dataset with 10% withheld for each of validation and testing. Training and testing on the same group of toy instances, this measures the ability to generalize to new angles, lighting conditions, backgrounds, and distortions.

We vary the amount of training data to plot learning curves in Figure 6.3. We show that CKMs are better able to generalize to these distortions than other methods, especially with less data. Importantly, the performance of **CKM** improves with more data, without

requiring costly optimization as data is added. We note that the benefit of $\mathbf{CKM}_W$ using weight learning becomes apparent with 200 training instances. This learning curve suggests that CKMs would be well suited for applications in cluttered environments with many 3D transformations (e.g., loop closure).

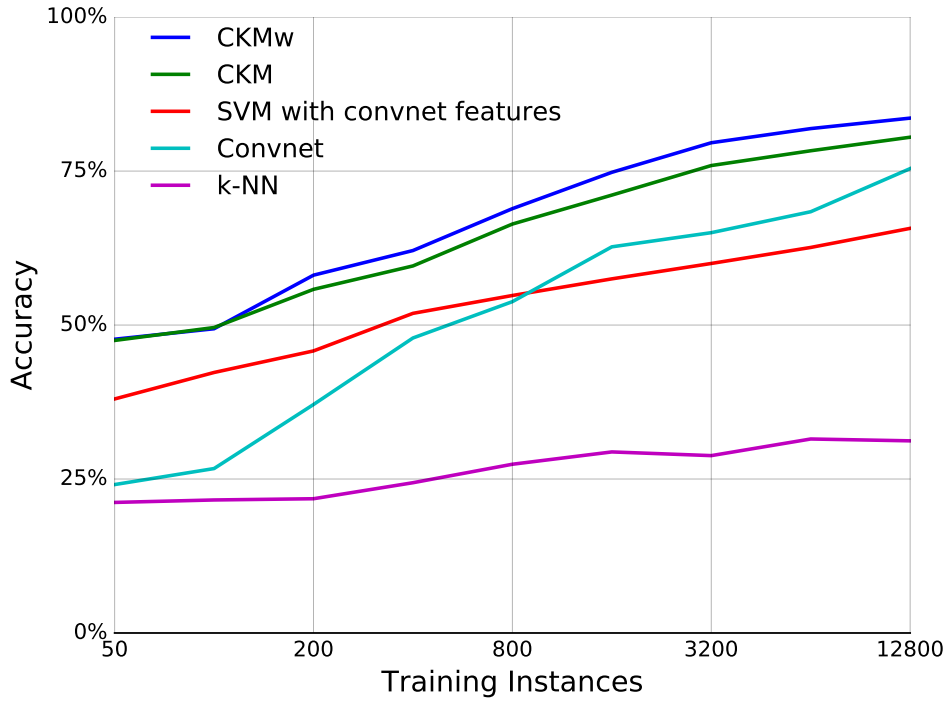

Figure 6.3: Number of training instances versus accuracy on unseen symmetries in NORB

6.3.5 Qualitative Comparison

We compare the salient properties of CKMs, SVMs, and convnets as used on these NORB variants in Table 6.4. The details for CKMs represent an unoptimized CPU implementation. For SVMs, we use the scikit-learn library [113]. Trained and pretrained convnets use the GPU-optimized TensorFlow library [1].

6.4 Conclusion

This chapter proposed compositional kernel machines, an instance-based method for object recognition that addresses some of the weaknesses of deep architectures and other kernel methods. We showed how using a sum-product function to represent a discriminant function leads to tractable summation over the weighted kernels to an exponential set of virtual instances, which can mitigate the curse of dimensionality and improve sample complexity. We proposed a method to discriminatively learn weights on individual instance elements and showed that this improves upon uniform weighting. Finally, we presented results in several scenarios showing that CKMs are a significant improvement for IBL and show promise compared with deep methods.

Table 6.4: System properties on NORB variants

Property	**CKM**	**SVM**	**Convnet**
Training time	Single pass: ∼5ms per image for ORB feature extraction and storage. If optimizing weights, ∼90ms per image, few epochs (CPU).	∼43ms per image for convnet feature extraction, singleton minibatch (GPU). Weight optimization <4ms per image added over all epochs (CPU).	∼2ms per image, minibatch 100 (GPU), many epochs.
Test time	∼80ms per image (CPU)	$<10\mu s$ per image (CPU)	∼1ms per image (GPU)
Memory cost	∼60K per training image, incl. optimization overhead (CPU)	>4G for pretrained convnet (GPU). <1M for classifier (CPU)	>4G for convnet (GPU)
Accuracy	Promising	Worse than CKM	State of the art with large datasets or pretrained networks
Use with small data	Can achieve high accuracy without extra data	Requires pretrained features	Requires pretrained features
Visual phenomena captured	Symmetries and composition	Some symmetries through kernels or virtual support vectors but no composition	Translation symmetry and composition
Difficulty for practitioners	Virtual instances, leaf kernels, and cost functions more intuitive than convnet architecture. Fast training. No data augmentation.	Choice of kernel. Optimization is efficient but opaque.	Many hyperparameters that indirectly describe visual representation. Lengthy training. Usually requires data augmentation.

Chapter 7

CONCLUSION

This dissertation has made several contributions to learning robust tractable models for vision. We have greatly expanded the applicability and usefulness of tractable probabilistic models by proposing discriminative parameter and generative structure learning algorithms for sum-product networks. We have also improved the robustness of neural networks to visual transformations by introducing deep symmetry networks. Finally, we have addressed weaknesses of learning deep visual models by proposing the tractable instance-based compositional kernel machine. In this chapter we summarize these contributions and suggest promising directions for future work.

7.1 Contributions of this Dissertation

We introduced the first algorithms for learning SPNs discriminatively, using a form of back-propagation to compute gradients. This work brings together probabilistic graphical models, deep networks, and discriminative training in a tractable model. We greatly expanded the types of SPN architectures by allowing evidence variables and their feature functions. We proposed both "soft" and "hard" gradient algorithms, using the latter to successfully combat the diffusion problem, allowing deep SPNs to be learned. Finally, we carried out experiments on image classification benchmarks showing that discriminative SPNs are competitive with deep networks.

We proposed a simple schema for learning SPN structure from data. Our algorithm recursively splits an SPN into a product of SPNs over independent sets of variables, if they can be found, or into a sum of SPNs learned from subsets of the instances, otherwise. We conducted experiments on a large number of datasets, showing that learned SPNs are

typically comparable in likelihood to graphical models but with superior inference speed and accuracy. This work has spurred much follow-on research on structure learning for sum-product networks and related models, including large-scale applications of probabilistic modeling that were previously intractable or over-simplified [82]. The basic schema of the LearnSPN algorithm has also led to generalized sum-product function algorithms for domains such as optimization [54].

We introduced deep symmetry networks, the first deep architecture that can compute features over any Lie group. It is a natural generalization of convolutional neural networks with feature maps that represent the application of a filter with any transformation and pooling to provide invariance to those transformations. Symnets use kernel interpolation and transformation optimization to address the difficulties in representing high-dimensional feature maps. On two image datasets with 2D and 3D variability we showed that affine symnets achieve higher accuracy than convnets while using significantly less data.

We proposed compositional kernel machines, an instance-based method for object recognition that addresses some of the weaknesses of deep architectures and other kernel methods. We introduced the concept of an exponential set of virtual instances and showed how they can mitigate the curse of dimensionality. We showed how to compactly represent the weighted sum over kernels with the virtual instances using a sum-product function. We demonstrated a CKM architecture over image features and showed how to optimize weights over individual image elements. Finally, we presented results showing that CKMs are a significant improvement over standard instance-based techniques and can be superior to deep methods on tasks requiring composition and symmetry. This work unifies many of the threads of this dissertation: tractability through sum-product decomposition, robustness to symmetries, and discriminative learning.

7.2 Directions for future research

7.2.1 Extensions of individual projects

This thesis enables several important applications of SPNs. First, discriminative learning of SPNs for structured prediction allows for exact inference over richer label dependencies than currently possible with conditional random fields. Second, the "hard" inference developed in Chapter 3 could give temporal SPNs an advantage modeling long-term dependence, benefiting applications in language, robotics, or any time-series data. Finally, the fast learning algorithms featured in Chapter 6 and Peharz et al. [114] could make SPNs suitable for real-time applications.

Deep symmetry networks suggest new research avenues. It would be fruitful to extend these networks to symmetries of 3D space. Symmetries of light and material are especially important for robotics, as a large number of objects are specular or refractive. Since real scenes have perceptual aliasing, it is important to learn symmetries in an interactive setting so that models can act to resolve ambiguity.

Compositional kernel machines are a promising representation with many possible architectures and learning procedures. For example, it would be beneficial to learn higher-order weights and cost functions as well as hidden variables. CKMs could be applied to structured prediction, regression, and reinforcement learning problems in AI such as natural language understanding, robot control, and games. CKMs exhibit a reversed trade-off of fast learning speed and large model size compared to neural networks. Given that animals can benefit from both trade-offs, these results may inspire computational theories of different brain structures, especially the neocortex versus the cerebellum [74].

7.2.2 Automated vision system

The strengths of the projects in this dissertation are complementary, and so it is important to describe why and how they might be combined into an automated vision system. We have shown the benefits of tractable inference and summation using the sum-product decomposi-

tion, but it could be a more robust visual representation if it could model symmetries other than translation. Likewise, the transformation optimization in Deep Symmetry Networks would perform better if the filters in the network were pieces of real images (i.e. part of a CKM) instead of randomly-initialized weights in a neural network. Here we provide suggestions and design considerations for future research that would merge CKMs and Symnets. In principle, such a system could have realtime inference and learning while not requiring expensive data augmentation to handle common visual transformations.

The first challenge is to add symmetries to CKMs with inference that scales to large datasets. The representation of leaf kernels and operators in a CKM should be augmented so that they compute both a value and a coordinate in the symmetry group. Product nodes still compute the scores of conjoined regions of the query image, but sum nodes can now weigh a child product based on the coordinates of its multiplicands. This would penalize a composition of image regions with transformations very different from what has been seen before (e.g., a left eye that is much larger or rotated from a right eye). Given the multiplicity and arbitrary detection of leaf kernels in a symmetry space, the CKM can have static or dynamic architecture. As with typical convnets, a static architecture would predefine the coordinates of sum nodes, each with a region of symmetry space that it pools over. Like a symnet, a dynamic CKM architecture would define the value of a sum node at all coordinates. The advantage of the static architecture is that computation is bounded; the disadvantage is that it may be costly to prearrange coordinates (beyond translation) that are adequate for all visual concepts. In theory, a dynamic architecture can align to any image, but it cannot be exhaustively evaluated and requires sampling or other approximation. Though the control points in symnets are dynamic, their initialization had the drawback of being prearranged. It would be prudent to explore bottom-up proposals of dynamic coordinates using techniques like mean shift [25] or nonparametric belief propagation [145].

The second challenge is creating a CKM learning algorithm and implementation that can scale to very large vision datasets. An ideal algorithm would learn a sparse set of weights in an online fashion. The original SPN paper used an L_0 penalty for this purpose [120].

Though uniform weight initialization performed well in that paper and in Chapter 6, it is less feasible for larger datasets. Alternatives include initialization of weights to zero and uniform for a random subset. Another important consideration is the semiring. Though the max-product and max-sum semirings have provided good results in SPN and CKM projects, the sparse gradient of the children of a max node can hinder optimization. The log (softmax) semiring is a promising alternative. Beyond sparse weights, an efficient implementation is necessary for very large or streaming datasets. For the evaluation of leaf kernels, cascade techniques (e.g., Simard et al. [138]) may provide better performance on parallel hardware than the k-means tree used in Chapter 6. Evaluation of product nodes in the CKM could be accelerated with techniques such as A* [64].

This research into robust tractable models hopefully lays the groundwork for automated vision systems of the future. Optimistically, the principles of sum-product decomposition, discriminative training, structure learning, and symmetry-based learning could even aid in understanding the tremendous visual computations so effortlessly performed by people and animals.

BIBLIOGRAPHY

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Yaser S Abu-Mostafa. Hints and the VC dimension. *Neural Computation*, 5(2):278–288, 1993.

[3] David Ackley, Geoffrey Hinton, and Terrence Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[4] Galen Andrew and Jianfeng Gao. Scalable training of L1-regularized log-linear models. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, 2007.

[5] Fabio Anselmi, Joel Z Leibo, Lorenzo Rosasco, Jim Mutch, Andrea Tacchetti, and Tomaso Poggio. Unsupervised learning of invariant representations in hierarchical architectures. ArXiv preprint 1311.4158, 2013.

[6] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[7] Francis R. Bach and Michael I. Jordan. Thin junction trees. *Advances in Neural Information Processing Systems 15*, 14:569–576, 2002.

[8] Simon Baker and Iain Matthews. Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[9] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[10] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, 34(5), 2007.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[12] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[13] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference*, 2010.

[14] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Kernel descriptors for visual recognition. *Advances in Neural Information Processing Systems*, 2010.

[15] Liefeng Bo, Kevin Lai, Xiaofeng Ren, and Dieter Fox. Object recognition with hierarchical kernel descriptors. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 1729–1736. IEEE, 2011.

[16] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Unsupervised feature learning for RGB-D based object recognition. In *Experimental Robotics*, pages 387–402. Springer, 2013.

[17] Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 1–8. IEEE, 2008.

[18] Antoine Bordes, Léon Bottou, Patrick Gallinari, and Jason Weston. Solving multiclass support vector machines with LaRank. In *Proceedings of the 24th International Conference on Machine Learning*, pages 89–96. ACM, 2007.

[19] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992.

[20] Christoph Bregler and Stephen M Omohundro. *Surface learning with applications to lipreading*. International Computer Science Institute, 1994.

[21] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

[22] Peter J Burt and Edward H Adelson. The Laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, 31(4):532–540, 1983.

[23] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

[24] Anton Chechetka and Carlos Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems 21*, pages 273–280. 2008.

[25] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[26] David Maxwell Chickering. The WinMine Toolkit. *Microsoft, Redmond, WA MSR-TR-2002-103*, 2002.

[27] David Maxwell Chickering, David Heckerman, and Christopher Meek. A Bayesian approach to learning Bayesian networks with local structure. In *UAI 13*, pages 80–89, 1997.

[28] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 3642–3649, 2012.

[29] Adam Coates and Andrew Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *International Conference on Machine Learning*, volume 8, page 10, 2011.

[30] Adam Coates and Andrew Y. Ng. Selecting receptive fields in deep networks. In *Advances in Neural Information Processing Systems 24*, pages 2528–2536, 2011.

[31] Adam Coates, Honglak Lee, and Andrew Y Ng. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the 15th Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 215–223. Society for Artificial Intelligence and Statistics, 2011.

[32] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, Philadelphia, PA, 2002. ACL.

[33] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[34] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20 (3):273–297, 1995.

[35] Thomas M Cover and Peter E Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[36] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(Dec):265–292, 2001.

[37] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, volume 1, pages 886–893. IEEE, 2005.

[38] Adnan Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50:280–305, 2003.

[39] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[40] Jesse Davis and Pedro Domingos. Bottom-up learning of Markov network structure. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, pages 271–280, 2010.

[41] Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.

[42] Dennis Decoste and Bernhard Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002.

[43] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19: 380–392, 1997.

[44] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[45] Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems 25*, pages 2042–2050, 2012.

[46] Persi Diaconis. *Group representations in probability and statistics*. Institute of Mathematical Statistics, 1988.

[47] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems 27*, pages 766–774, 2014.

[48] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3D object recognition. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 998–1005. IEEE, 2010.

[49] Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification*. John Wiley & Sons, 2000.

[50] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[51] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient matching of pictorial structures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 66–73. IEEE, 2000.

[52] Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, (1):67–92, 1973.

[53] Abram L Friesen and Pedro Domingos. Recursive decomposition for nonconvex optimization. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015.

[54] Abram L Friesen and Pedro Domingos. The sum-product theorem: A foundation for learning tractable models. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[55] King Sun Fu. *Syntactic Methods in Pattern Recognition*, volume 112. Elsevier, 1974.

[56] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[57] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems 25*, pages 3248–3256, 2012.

[58] Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 873–880, 2013.

[59] Vibhav Gogate, William Webb, and Pedro Domingos. Learning efficient Markov networks. In *Advances in Neural Information Processing Systems 23*, pages 748–756, 2010.

[60] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. *Proceedings of the Thirtieth International Conference on Machine Learning*, 28:1319–1327, 2013.

[61] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2015.

[62] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1458–1465. IEEE, 2005.

[63] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[64] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[66] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2957–2964. IEEE, 2012.

[67] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[68] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[69] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *Proceedings of the Twenty-First International Conference on Artificial Neural Networks*, 2011.

[70] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 1991.

[71] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Pro-*

ceedings of the 25th International Conference on Machine Learning, pages 408–415. ACM, 2008.

[72] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1): 106, 1962.

[73] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[74] Masao Ito. *The Cerebellum: Brain for an Implicit Self*. FT press, 2012.

[75] Tommi Jaakkola, David Haussler, et al. Exploiting generative models in discriminative classifiers. *Advances in Neural Information Processing Systems*, pages 487–493, 1999.

[76] Yangqing Jia, Chang Huang, and Trevor Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *Computer Vision and Pattern Recognition (CVPR), 2012. IEEE Conference on*, 2012.

[77] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[78] Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD)*, pages 217–226. ACM, 2006.

[79] Bela Julesz. Texton gradients: The texton theory revisited. *Biological Cybernetics*, 54 (4-5):245–251, 1986.

[80] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[81] Imre Risi Kondor. *Group Theoretical Methods in Machine Learning*. Columbia University, 2008.

[82] Viktoriya Krakovna and Moshe Looks. A minimalistic approach to sum-product network learning for real applications. In *International Conference on Learning Representations 2016 Workshop Track*, 2016.

[83] Balaji Krishnapuram, Lawrence Carin, Mario AT Figueiredo, and Alexander J Hartemink. Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27 (6):957–968, 2005.

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, 2012.

[85] Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. *Advances in Neural Information Processing Systems*, 20:785–792, 2007.

[86] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. ArXiv preprint 1207.6083, 2012.

[87] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, Williamstown, MA, 2001. Morgan Kaufmann.

[88] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, 2007.

[89] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[90] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[91] 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.

[92] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 2004.

[93] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting Structured Data*, 1:0, 2006.

[94] Mary S Lee and AW Moore. Efficient algorithms for minimizing cross validation error. In *Proceedings of the 8th International Conference on Machine Learning*, page 190. Morgan Kaufmann, 1994.

[95] Taehee Lee and Stefano Soatto. Video-based descriptors for object recognition. *Image and Vision Computing*, 29(10):639–652, 2011.

[96] Daniel Lowd. The Libra Toolkit. URL `http://libra.cs.uoregon.edu/`. Version 0.5.0, 2012.

[97] Daniel Lowd and Jesse Davis. Learning Markov network structure with decision trees. In *Proceedings of the Tenth IEEE International Conference on Data Mining*, pages 334–343. IEEE, 2010.

[98] Daniel Lowd and Pedro Domingos. Learning arithmetic circuits. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, Helsinki, Finland, 2008. AUAI Press.

[99] David G Lowe. Object recognition from local scale-invariant features. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, 1999.

[100] Feng Lu and Evangelos Milios. Robot pose estimation in unknown environments by matching 2D range scans. *Journal of Intelligent and Robotic Systems*, 18(3):249–275, 1997.

[101] Aleksandr Luntz and Viktor Brailovsky. On estimation of characters obtained in statistical procedure of recognition. *Technicheskaya Kibernetica*, 3(6):6–12, 1969.

[102] Sancho McCann and David G Lowe. Local naive Bayes nearest neighbor for image classification. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 3650–3656. IEEE, 2012.

[103] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[104] Willard Miller. *Symmetry Groups and Their Applications*. Academic Press, 1972.

[105] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

[106] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application (VISSAPP)*, pages 331–340, 2009.

[107] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy Belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[108] R.M. Neal and G.E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *NATO ASI SERIES D: Behavioural and Social Sciences*, 89:355–370, 1998.

[109] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.

[110] Mathias Niepert. Markov chains on orbits of permutation groups. In *Proceedings of the Twenty-Eight Conference on Uncertainty in Artificial Intelligence*, 2012.

[111] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[112] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, 1988.

[113] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincnet Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[114] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *Learning Tractable Probabilistic Models Workshop*, 2014.

[115] Alex Pentland, Baback Moghaddam, and Thad Starner. View-based and modular eigenspaces for face recognition. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 84–91. IEEE, 1994.

[116] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the Fisher kernel for large-scale image classification. In *Proceedings of the Eleventh European Conference on Computer Vision (ECCV)*, pages 143–156. Springer, 2010.

[117] John C Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, January 1998.

[118] John C Platt and Timothy P Allen. A neural network classifier for the I1000 OCR chip. In *Advances in Neural Information Processing Systems 9*, pages 938–944, 1996.

[119] Tomaso Poggio. On optimal nonlinear associative recall. *Biological Cybernetics*, 19(4): 201–209, 1975.

[120] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 337–346, 2011.

[121] Maurice Queyranne. Minimizing symmetric submodular functions. *Mathematical Programming*, 82(1):3–12, 1998.

[122] Marc'Aurelio Ranzato and Geoffrey E. Hinton. Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 2551–2558. IEEE, 2010.

[123] Pradeep Ravikumar, Martin J. Wainwright, and John D Lafferty. High-dimensional ising model selection using L1-regularized logistic regression. *The Annals of Statistics*, 38(3):1287–1319, 2010.

[124] Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *STOC 36*, 2004.

[125] Amirmohammad Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of the Thirty-First International Conference on Machine Learning*, pages 710–718. Omnipress, 2014.

[126] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[127] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82: 273–302, 1996.

[128] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[129] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571. IEEE, 2011.

[130] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986.

[131] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep Boltzmann machines. In *Proceedings of the 12th Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 448–455. Society for Artificial Intelligence and Statistics, 2009.

[132] Jarkko Salojärvi, Kai Puolamäki, and Samuel Kaski. Expectation maximization algorithms for conditional likelihoods. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 752–759. ACM, 2005.

[133] Cordelia Schmid and Roger Mohr. Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–534, 1997.

[134] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[135] Bernhard Schölkopf, Chris Burges, and Vladimir Vapnik. Incorporating invariances in support vector learning machines. In *Artificial Neural Networks (ICANN)*, pages 47–52. Springer, 1996.

[136] Bernhard Schölkopf, Patrice Simard, Alex J Smola, and Vladimir Vapnik. Prior knowledge in support vector kernels. *Advances in Neural Information Processing Systems*, pages 640–646, 1998.

[137] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *Mathematical Programming*, 127(1): 3–30, 2011.

[138] Patrice Simard, Yann LeCun, and John S Denker. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems 5*, 1992.

[139] Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the 7th International Conference on Document Analysis and Recognition*, pages 958–963. IEEE, 2003.

[140] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[141] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America. A, Optics and image science*, 4(3):519–524, 1987.

[142] Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477. IEEE, 2003.

[143] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281, 1986.

[144] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2015.

[145] Erik B Sudderth, Alexander T Ihler, Michael Isard, William T Freeman, and Alan S Willsky. Nonparametric belief propagation. *Communications of the ACM*, 53(10): 95–103, 2010.

[146] Kah-Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20 (1):39–51, 1998.

[147] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems 26*, pages 2004–2012, 2013.

[148] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[149] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2014.

[150] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[151] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of

energy minimization methods for Markov random fields. In *Proceedings of the Ninth European Conference on Computer Vision (ECCV)*, pages 16–29. Springer, 2006.

[152] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[153] Yuandong Tian and Srinivasa G Narasimhan. Hierarchical data-driven descent for efficient optimal deformation estimation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2288–2295. IEEE, 2013.

[154] Radu Timofte, Tinne Tuytelaars, and Luc Van Gool. Naive Bayes image classification: beyond nearest neighbors. In *Proceedings of the Eleventh Asian Conference on Computer Vision (ACCV)*, pages 689–703. Springer, 2013.

[155] Michael E Tipping. Sparse Bayesian learning and the relevance vector machine. *The Journal of Machine Learning Research*, 1:211–244, 2001.

[156] Antonio Torralba, Rob Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 1–8. IEEE, 2008.

[157] Matthew Turk, Alex P Pentland, et al. Face recognition using eigenfaces. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–591. IEEE, 1991.

[158] Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1148–1154. AAAI Press, 2012.

[159] Vladimir Naumovich Vapnik. *Statistical Learning Theory*, volume 1. Wiley New York, 1998.

[160] Vladimir Naumovich Vapnik and Samuel Kotz. *Estimation of Dependences Based on Empirical Data*, volume 40. Springer-verlag New York, 1982.

[161] Martin J. Wainwright, Tommi S. Jaakkola, and Alan S. Willsky. Tree-reweighted belief propagation algorithms and approximate ML estimation by pseudo-moment matching. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

[162] Christian Wallraven, Barbara Caputo, and Arnulf Graf. Recognition with local features: the kernel recipe. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV)*, pages 257–264. IEEE, 2003.

[163] Paul Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, Cambridge, MA, 1975.

[164] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, (3):408–421, 1972.

[165] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

[166] Barnet Woolf. The log likelihood ratio test (the G-test). *Annals of Human Genetics*, 21(4):397–409, 1957.

[167] Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

VITA

Robert Gens is from Newton, Massachusetts. He received a Bachelor of Science in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology in 2009, a Master of Science in Computer Science and Engineering from the University of Washington in 2012, and a Doctor of Philosophy in Computer Science and Engineering from the University of Washington in 2016.