The Sum-Product Theorem and its Applications

Abram L. Friesen

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

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Pedro Domingos, Chair

Carlos Guestrin

Henry Kautz

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science and Engineering

University of Washington

Abstract

The Sum-Product Theorem and its Applications

Abram L. Friesen

Chair of the Supervisory Committee:
Professor Pedro Domingos
Paul G. Allen School of Computer Science and Engineering

Models in artificial intelligence (AI) and machine learning (ML) must be expressive enough to accurately capture the state of the world, but tractable enough that reasoning and inference within them is feasible. However, many standard models are incapable of capturing sufficiently complex phenomena when constrained to be tractable. In this dissertation, I study the cause of this inexpressiveness and its relationship to inference complexity. I use the resulting insights to develop more efficient and expressive models and algorithms for many problems in AI and ML, including nonconvex optimization, computer vision, and deep learning.

I first identify and prove the sum-product theorem, which states that in any semiring for inference to be tractable it suffices that the factors of every product have disjoint scopes; i.e., that they are *decomposable*. I show that this simple condition unifies and extends many results in the literature and enables the definition of highly-expressive model classes that are tractable and learnable for many of the most important problems in AI and ML. Second, I develop RDIS, a novel nonconvex optimization algorithm based on the sum-product theorem. I show both analytically and empirically that RDIS can be exponentially faster than standard approaches because it finds and exploits local decomposability. Third, I combine decomposability with submodularity to define submodular field grammars (SFGs), a novel class of probabilistic models that extends both sum-product networks and submodular

Markov random fields. SFGs define a novel stochastic image grammar in which each object in the grammar can have arbitrary region shape but in which approximate MAP inference remains tractable, the first image grammar formulation in which this is possible. Finally, I demonstrate the applicability of decomposability to deep learning. I present feasible target propagation, a novel algorithm for learning deep neural networks with hard-threshold activations – which cannot be trained with standard backpropagation-based methods – that learns more accurate models than competing methods.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This dissertation and the work contained in it would not have been possible without the help and support of a large number of people in my life. I am grateful to everyone who has assisted me, explicitly or implicitly, in reaching this goal.

I would first like to thank my advisor, Pedro Domingos, for giving me so much of his time and patience, and for providing a constant source of optimism and encouragement. Pedro has taught me many things about research and writing, but most importantly he taught me how to take an ambitious idea, transform that idea into a viable and meaningful research question, and then actually make progress on answering that question.

I would also like to thank David Wingate and Rajesh Rao for advising me during the earlier years of my PhD. Despite his myriad other responsibilities, David Wingate was more than willing to advise me during my visit to MIT and well afterwards. David is a brilliant researcher, a loving father, and one of the kindest people I've ever known. Raj was my first advisor when I started my PhD and gave me the freedom and encouragement to explore my ever-changing research interests.

My committee members, Carlos Guestrin, Henry Kautz, and Jeff Bilmes, are some of the most capable, insightful, and intelligent researchers that I know. I am thankful for their sage wisdom and critical but constructive feedback.

During my PhD, I've had the pleasure of working on a number of projects with a variety of knowledgeable and interesting people. I'd like to thank my co-authors and collaborators David Wingate, Yanping Huang, Mike Chung, Rajesh Rao, Andrew Meltzoff, Dieter Fox, Joseph Austerweil, Tom Griffiths, Kristi Morton, Magda Balazinska, and Pedro Domingos. I would also like to thank Josh Tenenbaum and Leslie Kaelbling for providing the funding for

my time at MIT, welcoming me into their groups, and sharing their considerable wisdom.

Despite not working directly with them, there are many people who have lent a hand and an ear, and for that I am thankful and glad to count them as friends. Andrzej Pronobis, Mathias Niepert, Daniel Lowd, Byron Boots, Marc Deisenroth, and Tom Erez have all provided useful advice on life and research, helpful feedback on ideas and proposals, and good times at conferences. Luke Zettlemoyer's open door, willingness to listen, and kindness was particularly helpful when I was changing research areas. Paris Koutris' knowledge of relational databases was invaluable when working on applications of the sum-product theorem to the relational semiring. With both Chloé Kiddon and Cyrus Rashtchian, I've enjoyed (dinner) parties and other extracurricular events, and then they've been there the next day to read a draft of a paper or attend a practice talk. Mark Yatskar and Nicholas FitzGerald have also provided much helpful feedback on my work.

Throughout my many years in graduate school, I've shared a lab and an office with many wonderful people. I would particularly like to thank Rob Gens for his insights, clarity of thought, and willingness to read drafts of most of my papers and provide feedback. I am tremendously grateful to Tony Fader and Morgan Dixon for being two of the most hilarious and laid back people I know. They've helped me take a step back and focus on the most important parts of life – namely, puns – on many an occasion. My officemate at MIT, Chris Baker, was another constant source of inspiration and laughter. Rahul Kidambi has always been willing and able to provide advice about optimization and encouragement for my current research direction. Jesse Dodge and Janara Christensen are great, happy, friendly people with whom I've enjoyed many conversations about research and life in general.

In my time here, the staff of UW CSE has consistently gone out of their way to improve my experience and smooth my path. Lindsay Michimoto and Elise deGoede Dorough are two of the nicest people in the world, and I was lucky to have them both as graduate advisors. I will be forever grateful for their kind words, support, and constant willingness

to help. Joel Cohn made sure I never had to deal with any of the tax, tuition, or other financial complications that arose during my PhD. Similarly, reimbursements and purchases were never a problem in Andrei Stabrovski's capable hands. Finally, Sophie Ostlund and Rebekah Hansen made my trips downstairs for coffee much more enjoyable with their puns and sharp wit.

Despite my lack of free time and stressy demeanor, I somehow managed to make (and retain) some amazing friends during my time at UW. Hopefully, I do not forget anyone but, if I do, I'm sorry and you're amazing. Within UW CSE, Kayur Patel is one of the most generous people I know, and has been a source of constant and unwavering support, despite me disappearing into research holes for weeks at a time. He's someone I can always turn to for advice or discussion about any topic, and has provided some of the deepest and most meaningful conversations I've had during my PhD. He also taught me to worry less about what others think and the value of working in coffee shops (I'd also thank the staff at Victrola on 15th for caffeinating me all these years). For many similar but also many different reasons, Anup Rao has been another one of my closest friends during this time, and we've also engaged in many challenging, interesting, and rewarding discussions. Beyond that, however, Anup has shown me that it's possible to take a more balanced and positive approach to life, even in the most stressful of times. He is a living embodiment of the phrase "dance like nobody is watching." Alex Jaffe and Nodira Khoussainova have, in many ways, very different approaches to life. And yet they share a joy for life and a commitment to their respective ways of living that has made them both wildly successful and impressive, and inspires me whenever I see them. Miro Enev and I grew up as researchers together at UW, but it has been our interactions outside of UW that have been the most memorable. Miro is simply one of the warmest, kindest, and most supportive people I know, and I am lucky to count him as a friend. While my and Yaw Anokwa's mentoring relationship was perhaps not the most productive of all time (through no great

fault of either, of course), that experience did ensure that him and his better half, Hélène Martin, became great friends and even better travel companions, for which I am grateful.

Outside of UW CSE, my longtime close friend Aaron Farmer has been steadfast in his support and encouragement, and has remained someone I can turn to for advice and help with decisions at any time. Aaron is a solid and important fixture in my life, and has provided me with many dinners, game nights, rides, and rooms. Him and Jesse Low have both provided many beers and belays, which have helped keep me sane and safe, and I look forward to climbing many more mountains with them. While I've seen Dave Murray less than I'd like over this time due to distance, I owe a large part of who I am today to his friendship and belief in me. No acknowledgments would be complete without mentioning the roaming band of detectives known as Deus Ex, who welcomed me into their group and made me part of something larger and more meaningful. For that, I must thank Eric Miles, Lee Granas, and David Morris (the Oban 4) as well as Kevin Miniter, Travis Kriplean, Eva Ringstrom, Paul Pham, Alex Jaffe, Nodira Khoussainova, Gena Barnabee, Anup Rao, Kayur Patel, and everyone else. I'd also like to thank the many others who have made my life better along the way, including Ethan Katz-Bassett, Ben Birnbaum, Genevieve Walker, Benjamin Ahlvin, Katie O'Leary, and the many friends I've made playing soccer and climbing.

The ability of PhD students to subsist and survive while in graduate school is made possible by fellowships and grants. My studies were supported by a fellowship from the National Sciences and Engineering Research Council of Canada (NSERC), funding from ONR and AFRL, and hardware from NVIDIA.

I am so thankful to my family for their support and encouragement during this journey. My parents, Trish and David, (somehow) ingrained in me the value of hard work and the ability and confidence to persist in the face of great difficulty, without which none of this would have been possible. Their respective partners, Barry and Linda, have also been there

for me during this time with coffee and conversation or dinner and a drink. I am lucky to have amazing siblings, Yascha and Phoebe, who are both brilliant and some of my favourite people to hang out with. Friesen Freestyles Forever. Finally, to my partner, Gena Barnabee, thank you for everything. Gena deserves far more credit and recognition than is possible to give here. She has been a constant and unceasing source of support, encouragement, patience, feedback, food, hugs, jokes, and much much more. She is surely the unsung heroine of this story, and I hope I can repay her love and kindness in the years to come.

Chapter 1

INTRODUCTION

The ability to construct a model from data and then reason within it is a core requirement of artificial intelligence (AI) and machine learning (ML), as seen in problems ranging from computer vision, such as image classification, scene parsing, and structure from motion, to computational biology, such as DNA sequencing and protein folding. For such a model to be useful, it has to be both expressive enough that it can accurately represent the data and tractable enough that reasoning or inference within the model is fast relative to the underlying task. Expressivity, however, comes at the cost of tractability: the more of the world that a model must accurately represent, the more computation is required to reason within that model. There thus exists an inherent tradeoff between the expressivity of a model and the tractability of inference within that model. This tradeoff is exacerbated when models are learned from data, as inference is a subroutine called at each iteration of learning. One method that has been advanced in the hope of improving this tradeoff is to use approximate instead of exact inference. Approximate inference can be useful in that it often provides a mechanism for trading off expressivity (in the form of accuracy) for computation at test time, but it is not a panacea as it does not in general improve tractability without sacrificing expressivity. Further, approximate inference interacts poorly with learning, often yielding poor and unreliable results [81]. Instead, this dissertation takes a different approach to improving this tradeoff, by first investigating inference from a computational and algorithmic perspective, and then using the resulting insights to develop novel models and algorithms that are more expressive and efficient than existing work. By better understanding the fundamental computational challenges of inference, new

types of tractable models and algorithms can be developed that address these challenges and thereby improve expressivity without sacrificing tractability.

Despite apparent differences, many of the most important and challenging problems in AI and ML, including probabilistic and logical inference, constraint satisfaction, nonconvex and combinatorial optimization, integration, and reasoning in (probabilistic) knowledge bases, all share a common structure. Namely, each of these problems consists of summing a function over a semiring (e.g., Aji and McEliece [4], Bacchus et al. [8], Bistarelli et al. [17], Dechter [39], Dechter and Mateescu [40], Green et al. [51], Wilson [149]). For example, in the Boolean semiring the sum and product semiring operations are disjunction and conjunction, and deciding satisfiability is summing a Boolean formula over all truth assignments. Similarly, computing the most-probable explanation (MPE) is summation over all states in the max-product semiring, etc. This common structure provides a powerful abstraction for better understanding inference in general, and can thus serve as a mechanism for developing novel algorithms and model that utilize and extend ideas from other seemingly-unrelated domains.

In particular, graphical models are a compact representation often used as a target for learning probabilistic models. However, inference in a graphical model takes time exponential in its treewidth [25], a common measure of complexity. Further, since inference is a subroutine of learning, graphical models are hard to learn unless restricted to those with low treewidth [9, 27], but few real-world problems exhibit this property. Recent research, however, has shown that probabilistic models can in fact be much more expressive than this while remaining tractable [43]. In particular, sum-product networks (SPNs) [48, 114] are a class of deep probabilistic models that consist of many layers of hidden variables and can have unbounded treewidth. Despite this, inference in SPNs is guaranteed to be tractable, and their structure and parameters can be effectively and accurately learned from data [47, 48, 119].

The main contributions of this dissertation begin in Chapter 3, in which I generalize and extend the ideas behind SPNs to any problem that consists of summing a function over

a semiring, thereby enabling learning tractable high-treewidth representations for a much wider class of problems, including satisfiability, MAX-SAT, model counting, constraint satisfaction, marginal and MPE inference, integration, nonconvex optimization, database querying, and first-order probabilistic inference. To achieve this, I first identify and prove the sum-product theorem, a unifying principle for tractable inference that states a simple sufficient condition for summation to be tractable in any semiring: that the factors of every product have disjoint scopes; i.e., that the function or model is *decomposable*. In "flat" representations like graphical models and conjunctive normal form, consisting of a single product of sums, decomposability would allow only trivial models; but in deep representations like SPNs and negation normal form it allows for remarkable expressivity, especially when combined with techniques like dynamic programming. In the remainder of Chapter 3, I explore the benefits of decomposability and the sum-product theorem as they apply to both inference and learning for each of the problems listed above. I show that a number of existing and novel results are corollaries of the sum-product theorem, which allows me to define multiple new tractable model classes. I demonstrate the power and generality of the sum-product theorem by applying it to a new type of structured-prediction problem: learning a nonconvex function that is easy to optimize. I show empirically that this greatly outperforms the standard approach of learning without regard to the cost of optimization.

A key reason that deep decomposable representations are more expressive than shallow ones is that deep representations enable locally-decomposable operations, meaning that products or sums within the function are relevant only for a specific subregion of the function's domain and thus need only be decomposable within that subregion. Conversely, globally-decomposable operations must be decomposable over the entire domain, a much more restrictive condition. In probabilistic modeling, for example, conditional independence is a type of global decomposability as it necessarily holds for every point in the domain, whereas context-specific independence [19] is a type of local decomposability since it only applies in a particular subdomain. Local decomposability thus provides a

mechanism for improving the expressivity-tractability tradeoff, as shown in Figure 1.1.

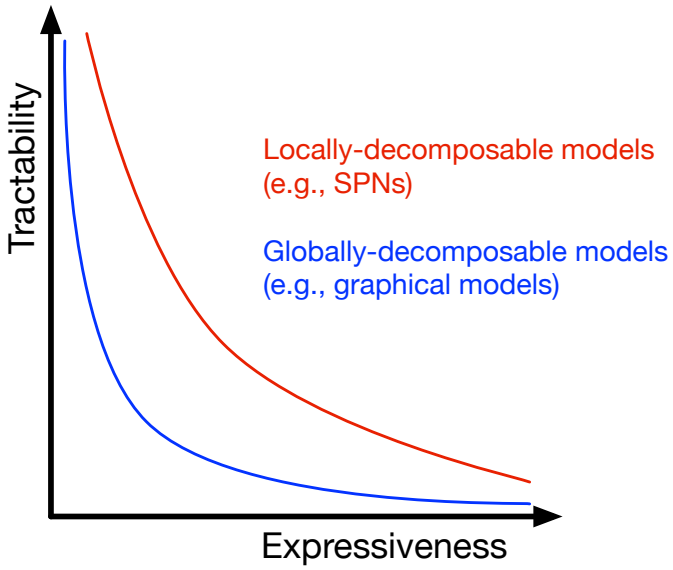


Figure 1.1: Expressivity-tractability tradeoff curves for locally- and globally-decomposable models.

Due to its prevalence and flexibility, local decomposability also provides an avenue for developing efficient inference algorithms for non-decomposable functions. In the same way that non-linear functions often exhibit local linearity despite not being globally linear, functions are often locally decomposable or approximately locally decomposable despite not being globally decomposable. In Chapter 4, I show that this is particularly beneficial for nonconvex optimization, where there are often an exponential number of local minima to explore, which makes optimization difficult; however, by explicitly finding and exploiting local decomposability, the number of minima to explore can be reduced by an exponential amount. Chapter 4 contributes a novel algorithm, RDIS, for nonconvex optimization that achieves this by recursively decomposing the given function into approximately independent subproblems in the current subregion of the space and then solving each of these separately. This enables RDIS to provably asymptotically find the global minimum of a broad class of nonconvex functions in exponentially less time than traditional methods. Empirically, I show that RDIS significantly outperforms standard nonconvex algorithms on

multiple challenging problems, including structure from motion and protein folding.

Decomposability is a powerful condition that defines a set of highly-expressive tractable model classes, including SPNs, that can efficiently model complex, high-level relationships between subparts of the input. Nonetheless, there remain problems that cannot be efficiently represented by an SPN, but for which tractable representations exist. Submodularity, for example, enables exponential reductions in inference complexity, and is commonly exploited to obtain tractable models for computer vision [21, 52, 75, 77], social networking [70, 105], and many other machine learning domains [16, 78]. The MAP state of a submodular Markov random field (MRF) can be computed in low-order polynomial time, despite having an exponential number of possible states. However, the same MRF is unable to tractably encode high-level relationships between arbitrary inputs. Conversely, an SPN would require exponential size to encode the same distribution as the MRF, but can efficiently model high-level relationships. In Chapter 5, I present submodular field grammars (SFGs), a novel probabilistic model that extends both submodular MRFs and SPNs, thereby combining the expressive power of both. An SFG can be interpreted as an SPN in which the weight of each sum-node child is given by the energy of a particular state of a submodular energy function. For clarity, SFGs are developed in terms of stochastic image grammars – a subset of the SPN model class – in Chapter 5, but the relationship between SPNs and SFGs is made clear in Section 5.3.1. By exploiting submodularity and decomposability, the approximate MAP state of an SFG can be computed in low-order polynomial time using SFG-Parse, a novel and provably convergent move-making algorithm for inference in SFGs. I show empirically that SFGs are a promising model for scene understanding and that inference with SFG-Parse takes exponentially less time than with standard algorithms while returning comparable results.

Beyond its utility for defining model classes that are tractable and easy to learn, decomposability can also be used to directly develop novel learning algorithms. Learning is commonly formulated as an optimization problem in which (stochastic) gradient descent is used to minimize a nonconvex loss over the training data, for example in deep learning where inference is already tractable. Unfortunately, RDIS is not immediately applicable

because the optimization variables (i.e., the model weights) in deep learning do not decompose locally. However, in Chapter 6, I show that it is possible to decompose the learning problem both across and within each layer of a deep network by specifying a set of targets for each hidden-layer activation. The resulting framework provides an effective approach for learning networks containing hard-threshold activations, which cannot be learned via standard backpropagation-based methods because the derivative of a hard-threshold activation is zero almost everywhere. Building on these ideas, I present feasible target propagation, a novel algorithm for training hard-threshold networks that includes the popular but poorly-justified straight-through estimator as a special case. Empirically, I show that feasible target propagation learns significantly more accurate deep hard-threshold networks than the straight-through estimator when applied to image classification.

Outside of my own work, the sum-product theorem has also proved useful for assisting the development of novel tractable models by others. For example, tractable probabilistic knowledge bases (TPKBs) [108, 146], which use tractable Markov logic [42] as the representation, define a highly-expressive class of first-order probabilistic models in which inference and learning can both be performed efficiently. However, because TPKBs combine SPNs and relational logic, the proof of their tractability is quite long. In Section 3.5.6, I show that this proof can be shortened significantly with the sum-product theorem, greatly simplifying their definition. In other work, compositional kernel machines (CKMs) [49] are a deep kernel method that incorporate the compositionality and symmetry of convolutional neural networks, resulting in a highly-expressive class of models that can learn from fewer samples and without data augmentation. To prove the tractability of CKMs, Gens and Domingos [49] explicitly formulate CKMs using the semiring-generalized framework that I present in Chapter 3 (and that also appeared in Friesen and Domingos [45]) and show that they are decomposable. The tractability of CKMs then follows immediately from the sum-product theorem.

In summary, Chapter 3 investigates inference from an abstract semiring-based perspec-

tive, proving the sum-product theorem and a number of corollaries that make clear that decomposability is a simple condition that defines tractable but expressive model classes that can be learned directly from data for a wide variety of core problems in AI and ML. Chapter 4 then uses these ideas to develop RDIS, a powerful nonconvex optimization algorithm that explicitly finds and exploits decomposability to achieve exponential improvements in optimization time. Chapter 5 shows that decomposability can be effectively combined with other types of structure, specifically submodularity, to define even more expressive models like SFGs, which extend both submodular MRFs and SPNs. Chapter 6 demonstrates that the benefits of decomposability extend beyond defining tractable and expressive model classes, and can be used to develop effective learning algorithms like feasible target propagation. Finally, Chapter 7 summarizes the contributions of this dissertation and suggests promising directions for future work. Background material is presented in Chapter 2.

Chapter 2

BACKGROUND

2.1 Graphical Models

A probabilistic graphical model is a pair (P, G) containing a graph $G = (V, E)$ with vertices V and edges E, and a probability distribution $P(\mathbf{X})$ over random variables $\mathbf{X} = (X_1, \ldots, X_n)$ corresponding to the n vertices of G. Each variable X_i takes values $x_i \in \mathcal{X}_i$ where $\mathcal{X}_i$ is the domain of X_i, which may be continuous (e.g., $\mathcal{X}_i \subseteq \mathbb{R}$) or discrete (e.g., $\mathcal{X}_i = \{1, \ldots, r\}$). The type and structure of the graph G specifies how the distribution P factorizes.

If the edges in G are undirected, then (P, G) is a Markov random field (MRF) and P factorizes according to the functions defined on the cliques $C \in \mathcal{C}$ of the graph, where a clique is a fully-connected subset of the variables. For an MRF, P can be written as

$$P(\mathbf{X}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{X}_C), \tag{2.1}$$

where each ψ_C is a potential over the clique variables $\mathbf{X}_C \subseteq \mathbf{X}$, and Z is known as the partition function [111]. The partition function $Z = \sum_{\mathbf{x} \in \mathcal{X}} \prod_{C \in \mathcal{C}} (\mathbf{x}|_C)$ is a constant chosen to ensure that the distribution is normalized, where $\mathbf{x}$ ranges over all settings of the variables $\mathbf{X}$ and $\mathbf{x}|_C$ denotes the values of subset $\mathbf{X}_C$ that are specified by $\mathbf{x}$.

Alternatively, (P, G) is a Bayesian network when the edges are directed and the graph is acyclic. In a Bayesian network, the potentials are conditional probabilities and P factorizes according to

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i | \mathbf{X}_{\text{pa}(i)}), \tag{2.2}$$

where $\mathbf{X}_{\text{pa}(i)}$ are the parents of X_i in the graph and each $P(X_i|\mathbf{X}_{\text{pa}(i)})$ denotes the conditional probability of X_i given its parents. The partition function is always equal to one when the potentials are conditional probabilities, and is thus omitted from the above equation.

Given a probabilistic model, computing the likelihood of observed data, computing the marginal distribution for some subset of the variables, and computing a conditional distribution are all easy given the ability to compute the probability of evidence $\mathbf{e} \in \mathcal{X}_E$ for a subset of the variables $\mathbf{X}_E \subseteq \mathbf{X}$, i.e.,

$$P(\mathbf{e}) = \sum_{\mathcal{X}_{\overline{E}}} P(\mathbf{e}, \mathbf{X}_{\overline{E}}), \tag{2.3}$$

where $\mathbf{X}_{\overline{E}} = \mathbf{X} \backslash \mathbf{X}_E$ and $\mathcal{X}_{\overline{E}}$ is the domain of $\mathbf{X}_{\overline{E}}$. The partition function is simply the unnormalized probability of empty evidence ($\mathbf{X}_E = \varnothing$). I thus refer to the task of computing the probability of some evidence as probabilistic inference.

Exact inference in a probabilistic graphical model can be performed with the junction tree algorithm [111], which first defines a new tree-structured graph (called a junction tree) from the original graph by moralizing the original graph, triangulating the moralized graph, defining a new graph from the maximum cliques of the triangulated graph, and finding a maximum spanning tree of this new graph. Given a junction tree, inference in the original graphical model can be performed by passing messages in a particular order on the graph. The treewidth $tw(\mathcal{T}) = \max_{C \in \mathcal{C}} |C| - 1$ of a junction tree $\mathcal{T}$, where $\mathcal{C}$ are the maximum cliques of the triangulated graph, determines the complexity of inference in the original model. In particular, inference in a junction tree takes best- and worst-case time that is exponential in the treewidth $tw(\mathcal{T})$ [25]. Thus, exact inference is only tractable for graphical models that have low-treewidth junction trees, such as chain graphs. Unfortunately, low-treewidth models are insufficiently expressive for most applications.

Alternatively, high-treewidth models can be used if inference is performed approximately. Approximate inference in graphical models can be performed with a variety of methods, including loopy belief propagation [147], variational techniques [145], and Monte

Carlo sampling methods [99]. However, even approximate inference can be NP-hard [122]. Further, the use of approximate inference methods – even those with guarantees – can reduce the expressivity of a model, making previously simple concepts impossible to implement, and can lead standard learning algorithms astray such that they never find valid model parameters [81].

Beyond probabilistic reasoning, graphical models can be used for other problems, including maximum a posteriori (MAP) inference and constraint satisfaction. Inference in each of these problems can be seen as a special case of the generalized distributive law [4], a general message-passing algorithm for models in which inference consists of summing a function over a semiring, where that function is a product of factors. More details on this relationship are provided in Chapter 3.

2.2 *Sum-Product Networks*

Sum-product networks (SPNs) are a deep but tractable class of probabilistic models. Formally, following Gens and Domingos [48], an SPN can be defined recursively, where the scope of an SPN is the set of variables that appear in it, and a tractable distribution is one in which its partition function and its mode can be computed in constant time.

Definition 2.1 (Gens and Domingos [48])**.** *A* sum-product network (*SPN*) *is any of the following*:

1. *A tractable univariate distribution.*
2. *A product of SPNs with disjoint scopes.*
3. *A weighted sum of SPNs with the same scope in which all weights are positive.*

A slightly more general definition of SPNs is given in Poon and Domingos [114], but the definition in Gens and Domingos [48] is simpler and suffices for this dissertation. Using this definition, an SPN is *decomposable*, since the children of each product have disjoint scopes, and *complete*, since the children of each sum have the same scope.

An SPN $S(\mathbf{X})$ over variables $\mathbf{X} = (X_1, \ldots, X_n)$ is typically represented as a rooted directed acyclic graph (DAG) with sums and products as the internal nodes, and univariate distributions over the variables $\mathbf{X}$ as the leaves. The edge from each sum node to each of its children is labeled with the corresponding weight of that child. The size of an SPN is the number of edges in its graph. An SPN has probabilistic semantics both for its root S and for each sub-SPN S_i rooted at node i, as each sub-SPN represents a probability distribution over its scope. The probability of some value $\mathbf{x} \in \mathcal{X}$ of the variables is $P(\mathbf{x}) = S(\mathbf{x})/Z$, where $Z = \sum_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x})$ is the partition function.

Intuitively, each sum node in an SPN can be interpreted as explicitly marginalizing out a latent mixture variable, where the values of the latent variables (i.e., the children of the sum node) in an SPN are often taken to specify subregions of the domain of the SPN. In this regard, each product node represents a particular context-specific independence [19] in the subregion of the domain defined by the child edges of the sum nodes on the path(s) from the product node to the root.

As in a probabilistic graphical model, probabilistic inference in an SPN reduces to computing the probability of evidence $\mathbf{e} \in \mathcal{X}_E$ for a subset of the variables $\mathbf{X}_E \subseteq \mathbf{X}$, i.e.,

$$P(\mathbf{e}) = \sum_{\mathcal{X}_{\overline{E}}} P(\mathbf{e}, \mathbf{X}_{\overline{E}}), \tag{2.4}$$

where $\mathbf{X}_{\overline{E}} = \mathbf{X} \backslash \mathbf{X}_E$. However, unlike in a graphical model, marginal inference in an SPN is guaranteed to be tractable. In particular, the unnormalized probability of evidence of an SPN can be computed in time linear in the size of the SPN [48, 114] by replacing each leaf function over an evidence variable $\phi_l \in \{\phi_l(X_j) : X_j \in \mathbf{X}_E\}$ with the constant $\phi_l(\mathbf{e}_j)$, replacing each leaf function over a non-evidence variable with the partition function of that leaf function, and then evaluating the SPN. The value produced by the root node is the unnormalized probability of evidence.

Despite being tractable, SPNs are also quite expressive and contain many existing models as special cases [114], including thin junction trees (i.e., graphical models with low

treewidth), latent tree models, many types of mixture models, and probabilistic context-free grammars with bounded recursion [68]. SPNs are also highly related to arithmetic circuits (ACs) [35], which were originally used as a target for compiling Bayesian networks to reduce inference complexity. In particular, ACs and SPNs over discrete variables can represent the same distributions, but the SPN representation is more efficient [119]. Finally, the expressivity of Bayesian networks can be increased by using algebraic decision diagrams to represent the conditional probability distributions, thus enabling them to represent the same set of distributions as SPNs, but again less efficiently than an SPN. Specifically, Zhao et al. [153] shows that any complete and consistent (a weaker requirement than decomposability, see Poon and Domingos [114]) SPN S can be converted to such a Bayesian network in time $O(n|S|^2)$, where n is the number of variables and $|S|$ is the size of the SPN. The resulting Bayesian network has size $O(n|S|^2)$.

Chapter 3

THE SUM-PRODUCT THEOREM: A FOUNDATION FOR LEARNING TRACTABLE MODELS

3.1 Introduction

In this chapter, I generalize and extend the ideas behind SPNs to enable learning tractable high-treewidth representations for a much wider class of problems, including satisfiability, MAX-SAT, model counting, constraint satisfaction, marginal and MPE inference, integration, nonconvex optimization, database querying, and first-order probabilistic inference. The class of problems I address can be viewed as generalizing structured prediction beyond combinatorial optimization [139], to include optimization for continuous models and others. Instead of approaching each domain individually, I build on a long line of work showing how, despite apparent differences, these problems in fact have much common structure (e.g., Aji and McEliece [4], Bacchus et al. [8], Bistarelli et al. [17], Dechter [39], Dechter and Mateescu [40], Green et al. [51], Wilson [149]), namely, that each consists of summing a function over a semiring.

I begin by identifying and proving the sum-product theorem, a unifying principle for tractable inference that states a simple sufficient condition for summation to be tractable in any semiring: that the factors of every product have disjoint scopes. In "flat" representations like graphical models and conjunctive normal form, consisting of a single product of sums, this would allow only trivial models, but in deep representations like SPNs and negation normal form it provides remarkable flexibility. Based on the sum-product theorem, I develop an algorithm for learning representations that satisfy this condition, thus guaranteeing that the learned functions are tractable yet expressive. I demonstrate the power and generality of my approach by applying it to a new type of

structured prediction problem: learning a nonconvex function that can be optimized in polynomial time. Empirically, I show that this greatly outperforms the standard approach of learning a continuous function without regard to the cost of optimizing it. I also show that a number of existing and novel results are corollaries of the sum-product theorem, propose a general algorithm for inference in any semiring, define novel tractable classes of constraint satisfaction problems, integrable and optimizable functions, and database queries, and present a much simpler proof of the tractability of tractable Markov logic.

3.2 The Sum-Product Theorem

I begin by introducing the notation used throughout this chapter and defining several important concepts. I denote a vector of variables by $\mathbf{X} = (X_1, \ldots, X_n)$ and its value by $\mathbf{x} = (x_1, \ldots, x_n)$ where $x_i \in \mathcal{X}_i$ for all i and $\mathcal{X}_i$ is the domain of X_i. I denote subsets (for simplicity, tuples are treated as sets) of variables as $\mathbf{X}_A, \mathbf{X}_a \subseteq \mathbf{X}$, where the domains $\mathcal{X}_A, \mathcal{X}_a$ are the Cartesian product of the domains of the variables in $\mathbf{X}_A, \mathbf{X}_a$, respectively. I denote (partial) assignments as $\mathbf{a} \in \mathcal{X}_A$ and restrictions of these to $\mathbf{X}_B \subset \mathbf{X}_A$ as $\mathbf{a}_B$. To indicate compatibility between $\mathbf{a} \in \mathcal{X}_A$ and $\mathbf{c} \in \mathcal{X}_C$ (i.e., that $\mathbf{a}_j = \mathbf{c}_j$ for all $X_j \in \mathbf{X}_A \cap \mathbf{X}_C$), I write $\mathbf{a} \sim \mathbf{c}$. The *scope* of a function is the set of variables it takes as input.

Definition 3.1. *A* commutative semiring $(R, \oplus, \otimes, 0, 1)$ *is a nonempty set* R *on which the operations of sum* ($\oplus$) *and product* ($\otimes$) *are defined and satisfy the following conditions*:

(*i*) $(R, \oplus)$ *and* $(R, \otimes)$ *are associative and commutative, with identity elements* $0, 1 \in R$ *such that* $0 \neq 1$, $a \oplus 0 = a$, *and* $a \otimes 1 = a$ *for all* $a \in R$;

(*ii*) $\otimes$ *distributes over* $\oplus$, *such that* $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ *for all* $a, b, c \in R$; *and*

(*iii*) 0 *is absorbing for* $\otimes$, *such that* $a \otimes 0 = 0$ *for all* $a \in R$.

I am interested in computing summations $\bigoplus_{\mathbf{x} \in \mathcal{X}} F(\mathbf{x})$, for $(R, \oplus, \otimes, 0, 1)$ a commutative semiring and $F : \mathcal{X} \to R$ a function on that semiring, with $\mathcal{X}$ a finite set (but see Section 3.5.4 for extensions to continuous variables). I refer to such a function as a sum-product function.

Definition 3.2. *A* sum-product function (*SPF*) *over* $(R, \mathbf{X}, \Phi)$*, where* R *is a semiring,* $\mathbf{X}$ *is a set of variables, and* Φ *is a set of constant* ($\phi_l \in R$) *and univariate functions* ($\phi_l : \mathcal{X}_j \to R$ *for* $X_j \in \mathbf{X}$)*, is any of the following*: (*i*) *a function* $\phi_l \in \Phi$, (*ii*) *a product of SPFs, or* (*iii*) *a sum of SPFs.*

An SPF $S(\mathbf{X})$ computes a mapping $S : \mathcal{X} \to R$ and can be represented by a rooted directed acyclic graph (DAG), where each leaf node is labeled with a function $\phi_l \in \Phi$ and each non-leaf node is labeled with either $\oplus$ or $\otimes$ and referred to as a sum or product node, respectively. Two SPFs are *compatible* iff they compute the same mapping; i.e., $S_1(\mathbf{x}) = S_2(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, where $S_1(\mathbf{X})$ and $S_2(\mathbf{X})$ are SPFs. The *size* of an SPF is the number of edges in the graph. The DAG rooted at each node $v \in S$ represents a sub-SPF $S_v : \mathcal{X}_v \to R$ for $\mathbf{X}_v \subseteq \mathbf{X}$. Notice that restricting the leaf functions ϕ_l to be univariate incurs no loss of generality because any mapping $\psi : \mathcal{X} \to R$ is compatible with the trivial SPF

$$F(\mathbf{X}) = \bigoplus_{\mathbf{x} \in \mathcal{X}} \Big(\psi(\mathbf{x}) \otimes \bigotimes_{i=1}^{n} [X_i = \mathbf{x}_i] \Big),$$

where the indicator function $[.]$ has value 1 when its argument is true, and 0 otherwise (recall that 0 and 1 are the semiring identity elements). SPFs are similar to arithmetic circuits [134], but the leaves of an SPF are functions instead of variables. Darwiche [35] used arithmetic circuits as a data structure to support inference in Bayesian networks over discrete variables. An important subclass of SPFs are those that are decomposable.

Definition 3.3. *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.*

Decomposability is a simple condition that defines a class of functions for which inference is tractable.

Theorem 3.1 (Sum-product theorem)**.** *Every decomposable SPF can be summed in time linear in its size.*

Proof. The proof is recursive, starting from the leaves of the SPF. Let $S(\mathbf{X})$ be a decomposable SPF on commutative semiring $(R, \oplus, \otimes, 0, 1)$. Every leaf node can be summed in constant

time, because each is labeled with either a constant or univariate function. Now, let $v \in S$ be a node, with $S_v(\mathbf{X}_v)$ the sub-SPF rooted at v and $Z_v = \bigoplus_{\mathcal{X}_v} S_v(\mathbf{X}_v)$ its summation. Let $\{c_i\}$ be the children of v for $c_i \in S$, with sub-SPFs $S_i(\mathbf{X}_i)$ for $\mathbf{X}_i \subseteq \mathbf{X}_v$ and summations Z_i. Let $\mathcal{X}_{v \setminus i}$ be the domain of variables $\mathbf{X}_v \setminus \mathbf{X}_i$.

If v is a sum node, then

$$\begin{aligned} Z_v &= \bigoplus_{\mathcal{X}_v} \bigoplus_{i} S_i(\mathbf{X}_i) \\ &= \bigoplus_{i} \bigoplus_{\mathcal{X}_v} S_i(\mathbf{X}_i) \\ &= \bigoplus_{i} \bigoplus_{\mathcal{X}_{v \setminus i}} \bigoplus_{\mathcal{X}_i} S_i(\mathbf{X}_i) \\ &= \bigoplus_{i} Z_i \otimes \Big(\bigoplus_{\mathcal{X}_{v \setminus i}} 1 \Big), \end{aligned}$$

and computing the summation Z_v for v a sum node simply requires computing the sum of the summations Z_i for each child c_i.

If v is a product node, then any two children c_i, c_j for $i, j \in \{1, \ldots, m\}$ have disjoint scopes, $\mathbf{X}_i \cap \mathbf{X}_j = \varnothing$, and

$$\begin{aligned} Z_v &= \bigoplus_{\mathcal{X}_v} \bigotimes_{i} S_i(\mathbf{X}_i) \\ &= \bigoplus_{\mathcal{X}_1} \bigoplus_{\mathcal{X}_{v \setminus 1}} \bigotimes_{i} S_i(\mathbf{X}_i) \\ &= \bigoplus_{\mathcal{X}_1} S_1(\mathbf{X}_1) \otimes \bigoplus_{\mathcal{X}_{v \setminus 1}} \bigotimes_{i=2}^{m} S_i(\mathbf{X}_i) \\ &= \bigotimes_{i} \bigoplus_{\mathcal{X}_i} S_i(\mathbf{X}_i) \\ &= \bigotimes_{i} Z_i, \end{aligned}$$

and computing the summation Z_v for v a product node simply requires computing the

product of the summations Z_i for each child c_i.

The above equations only require associativity and commutativity of $\oplus$ and associativity and distributivity of $\otimes$, which are properties of a semiring. Thus, any node can be summed over its domain in time linear in the number of its children, and S can be summed in time linear in its size with a single upward pass through the SPF. □

I have assumed here that $\bigoplus_{\mathcal{X}_{v\setminus i}} 1$ can be computed in constant time and that each leaf function can be evaluated in constant time, which is true for all semirings considered. I have also assumed that $a \oplus b$ and $a \otimes b$ take constant time for any elements a, b of semiring R, which is true for most common semirings. Section 3.2.1 provides further detail on these assumptions and discusses the effect on the sum-product theorem when they are relaxed.

The complexity of summation in an SPF can be related to other notions of complexity, such as treewidth, the most common and relevant complexity measure across the domains considered in this chapter. To define the treewidth of an SPF, I first define junction trees [4, 85] and a related class of SPFs.

Definition 3.4. *A* junction tree *over variables $\mathbf{X}$ is a tuple (T, Q), where T is a rooted tree, Q is a set of subsets of variables, each vertex $i \in T$ contains a subset of variables $\mathbf{C}_i \in Q$ such that $\cup_i \mathbf{C}_i = \mathbf{X}$, and for every pair of vertices $i, j \in T$ and for all $k \in T$ on the (unique) path from i to j, $\mathbf{C}_i \cap \mathbf{C}_j \subseteq \mathbf{C}_k$. The separator for an edge $(i, j) \in T$ is defined as $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$.*

A junction tree provides a schematic for constructing a specific type of decomposable SPF called a tree-like SPF (a semiring-generalized version of a construction from Darwiche [35]). Note that a tree-like SPF is not a tree, however, as many of its nodes have multiple parents.

Definition 3.5. *A* tree-like SPF *over variables $\mathbf{X}$ is constructed from a junction tree $\mathcal{T} = (T, Q)$ and functions $\{\psi_i(\mathbf{C}_i)\}$ where $\mathbf{C}_i \in Q$ and $i \in T$, and contains the following nodes*:

(*i*) *a node ϕ_{vt} with indicator $\phi_t(X_v) = [X_v = t]$ for each value $t \in \mathcal{X}_v$ of each variable $X_v \in \mathbf{X}$;*

(*ii*) *a (leaf) node a_i with value $\psi_i(\mathbf{c}_i)$ and a product node c_i for each value $\mathbf{c}_i \in \mathcal{X}_{\mathbf{C}_i}$ of each cluster $\mathbf{C}_i$;*

(*iii*) *a sum node* s_{ij} *for each value* $\mathbf{s}_{ij} \in \mathcal{X}_{\mathbf{S}_{ij}}$ *of each separator* $\mathbf{S}_{ij}$*; and*

(*iv*) *a single root sum node* s*.*

A product node c_j *and a sum node* s_{ij} *are compatible iff their corresponding values are compatible; i.e.,* $\mathbf{c}_j \sim \mathbf{s}_{ij}$*. The nodes are connected as follows. The children of the root* s *are all product nodes* c_r *for* r *the root of* T*. The children of product node* c_j *are all compatible sum nodes* s_{ij} *for each child* i *of* j*, the constant node* a_j *with value* $\psi_j(\mathbf{c}_j)$*, and all indicator nodes* ϕ_{vt} *such that* $X_v \in \mathbf{C}_j$*,* $t \sim \mathbf{c}_j$*, and* $X_v \notin \mathbf{C}_k$ *for* k *any node closer to the root of* $\mathcal{T}$ *than* j*. The children of sum node* s_{ij} *are the compatible product nodes* c_i *of child* i *of* j *connected by separator* $\mathbf{S}_{ij}$*.*

If S is a tree-like SPF with junction tree (T, Q), then it is not difficult to see both that S is decomposable, since the indicators for each variable all appear at the same level, and that each sum node s_{jk} computes

$$S_{s_{jk}}(\mathbf{S}_{jk}) = \bigoplus_{(\mathbf{c} \in \mathcal{X}_{\mathbf{C}_j}) \sim \mathbf{s}_{jk}} \left(\psi_j(\mathbf{c}) \otimes [\mathbf{C}_j = \mathbf{c}] \otimes \left(\bigotimes_{i \in \mathrm{Ch}(j)} S_{s_{ij}}(\mathbf{c}_{\mathbf{S}_{ij}}) \right) \right),$$

where the indicator children of c_j have been combined into $[\mathbf{C}_j = \mathbf{c}]$, $\mathrm{Ch}(j)$ are the children of j, and $i, j, k \in T$ with j the child of k. Further, $S(\mathbf{x}) = \bigotimes_{i \in T} \psi_i(\mathbf{x}_{\mathbf{C}_i})$ for any $\mathbf{x} \in \mathcal{X}$. Thus, tree-like SPFs provide a method for decomposing an SPF. For a tree-like SPF to be compatible with an SPF F, it cannot assert independencies that do not hold in F.

Definition 3.6. *Let* $F(\mathbf{U})$ *be an SPF over variables* $\mathbf{U}$ *with pairwise-disjoint subsets* $\mathbf{X}, \mathbf{Y}, \mathbf{W} \subseteq \mathbf{U}$*. Then* $\mathbf{X}$ *and* $\mathbf{Y}$ *are* conditionally independent in F *given* $\mathbf{W}$ *iff* $F(\mathbf{X}, \mathbf{Y}, \mathbf{w}) = F(\mathbf{X}, \mathbf{w}) \otimes F(\mathbf{Y}, \mathbf{w})$ *for all* $\mathbf{w} \in \mathcal{W}$*, where* $F(\mathbf{X}) = \bigoplus_{\mathcal{Y}} F(\mathbf{X}, \mathbf{Y})$ *for* $\{\mathbf{X}, \mathbf{Y}\}$ *a partition of* $\mathbf{U}$*.*

Similarly, a junction tree $\mathcal{T} = (T, Q)$ is incompatible with F if it asserts independencies that are not in F, where variables X and Y are conditionally independent in $\mathcal{T}$ given $\mathbf{W}$ if $\mathbf{W}$ separates X from Y. A set of variables $\mathbf{W}$ *separates* X and Y in $\mathcal{T}$ iff after removing all vertices $\{i \in T : \mathbf{C}_i \subseteq \mathbf{W}\}$ from T there is no pair of vertices $i, j \in T$ such that $X \in \mathbf{C}_i$, $Y \in \mathbf{C}_j$, and i, j are connected.

Inference complexity is commonly parameterized by *treewidth*, defined for a junction tree $\mathcal{T} = (T, Q)$ as the size of the largest cluster minus one; i.e., $tw(\mathcal{T}) = \max_{i \in T} |\mathbf{C}_i| - 1$. The treewidth $tw(S)$ of an SPF S is the minimum treewidth over all junction trees compatible with S. Notice that these definitions of junction tree and treewidth reduce to the standard ones [69]. If the treewidth of S is bounded (i.e., there exists some fixed constant $0 < k < \infty$ such that $tw(S) \leq k$) then inference in S is efficient because there must exist a compatible tree-like SPF that has bounded treewidth. Note that the trivial junction tree with only a single cluster is compatible with every SPF.

Corollary 3.1. *Every SPF with bounded treewidth can be summed in time linear in the cardinality of its scope.*

Proof. Let $\mathbf{X} = (X_1, \ldots, X_n)$, let $(R, \oplus, \otimes, 0, 1)$ be a commutative semiring, and let $F(\mathbf{X})$ be an SPF with bounded treewidth, such that $tw(F) = a$ for a some fixed constant where $0 < a < \infty$. Let $S(\mathbf{X})$ be a tree-like SPF that is compatible with F and has junction tree $\mathcal{T} = (T, Q)$ with treewidth $tw(\mathcal{T}) = a$. The size of the largest cluster in $\mathcal{T}$ is $\alpha = a + 1$. Let $m = |Q| \leq n$ and $d = |\mathcal{X}_v|$ for all $X_v \in \mathbf{X}$. Further, other than the root $s \in S$, there is a one-to-one correspondence between separator instantiations $\mathbf{s}_{ij} \in \mathcal{X}_{\mathbf{S}_{ij}}$ and sum nodes $s_{ij} \in S$, and between cluster instantiations $\mathbf{c}_j \in \mathcal{X}_{\mathbf{C}_j}$ and product nodes $c_j \in S$. Now, the number of edges in S can be obtained by counting the edges that correspond to each edge in T and summing over all edges in T, as follows. By construction, each edge $(j, k) \in T$ corresponds to the product nodes $\{c_k\}$; their children, which are the leaf nodes (indicators and constants) and the sum nodes $\{s_{jk}\}$; and the children of $\{s_{jk}\}$, which are the product nodes $\{c_j\}$. By definition, the $\{c_j\}$ have only a single parent, so there are $|\mathcal{X}_{\mathbf{C}_j}| \leq d^\alpha$ edges between $\{s_{jk}\}$ and $\{c_j\}$. Further, each c_k has only $|\mathbf{C}_k|+1$ leaf node children and $|\text{Ch}(k)|$ sum node children, so there are $|\mathcal{X}_{\mathbf{C}_k}|(|\mathbf{C}_k|+1)(|\text{Ch}(k)|) \leq d^\alpha(\alpha+1)(|\text{Ch}(k)|)$ edges between $\{c_k\}$ and $\{s_{jk}\}$. In addition, there are also $\mathcal{X}_{\mathbf{C}_r} = d^\alpha$ edges between the root $s \in S$ and the product nodes c_r. Thus, since T is a tree with $m-1$ edges, $\text{size}(S) \leq d^\alpha + \sum_{(j,k) \in T} 2d^\alpha(\alpha+1)(|\text{Ch}(k)|) = O(md^\alpha)$, which is $O(n)$. Since S is decomposable and has size $O(n)$, then, from the sum-product

theorem, S can be summed in time $O(n)$. Furthermore, S is compatible with F, so F can be summed in time $O(n)$, and the claim follows. □

For any SPF, tree-like SPFs are just one type of compatible SPF, one with size exponential in treewidth; however, there are many other compatible SPFs. In fact, there can be compatible (decomposable) SPFs that are exponentially smaller than any compatible tree-like SPF.

Corollary 3.2. *In every semiring, not every SPF that can be summed in time linear in the cardinality of its scope has bounded treewidth.*

Proof. By counterexample. Let $\mathbf{X} = (X_1, \ldots, X_n)$ be a vector of variables, let $(R, \oplus, \otimes, 0, 1)$ be a commutative semiring, and let $k = |\mathcal{X}_i|$ for all $X_i \in \mathbf{X}$. The SPF $F(\mathbf{X}) = \bigoplus_{j=1}^{r} \bigotimes_{i=1}^{n} \psi_{ji}(X_i)$ can be summed in time linear in n because F is decomposable and has size $r(n+1)$. At the same time, $F(\mathbf{X})$ has treewidth $n-1$ (i.e., unbounded) because there are no pairwise-disjoint subsets $\mathbf{A}, \mathbf{B}, \mathbf{C} \subseteq \mathbf{X}$ with domains $\mathcal{X}_A, \mathcal{X}_B, \mathcal{X}_C$ such that $\mathbf{A}$ and $\mathbf{B}$ are conditionally independent in F given $\mathbf{C}$, and thus the smallest junction tree compatible with $F(\mathbf{X})$ is a complete clique over $\mathbf{X}$. This can be seen as follows. Without loss of generality, let $\mathbf{A} \cup \mathbf{B}$ be the first m variables in $\mathbf{X}$, such that $\mathbf{A} \cup \mathbf{B} = (X_1, \ldots, X_m)$. For any $\mathbf{c} \in \mathcal{X}_C$,

$$\begin{aligned} F(\mathbf{A}, \mathbf{B}, \mathbf{c}) &\propto \bigoplus_{j=1}^{r} \bigotimes_{i \,:\, X_i \in \mathbf{A} \cup \mathbf{B}} \psi_{ji}(X_i) \\ &= \Big(\psi_{11}(X_1) \otimes \cdots \otimes \psi_{1m}(X_m)\Big) \;\oplus\; \ldots \;\oplus\; \Big(\psi_{r1}(X_1) \otimes \cdots \otimes \psi_{rm}(X_m)\Big). \end{aligned}$$

For $F(\mathbf{A}, \mathbf{B}, \mathbf{c})$ to factor, the terms in the right-hand side must have common factors; however, in general, each ψ_{ji} is different, so there are no such factors. Thus, $F(\mathbf{A}, \mathbf{B}, \mathbf{c}) \neq F(\mathbf{A}, \mathbf{c}) \otimes F(\mathbf{B}, \mathbf{c})$ for all $\mathbf{c} \in \mathcal{X}_C$, and there are no conditional independencies in F. □

Given existing work on tractable high-treewidth inference, it is perhaps surprising that the above results do not exist in the literature at this level of generality. Most relevant is the preliminary work of Kimmig et al. [72], which proposes a semiring generalization

of arithmetic circuits for knowledge compilation and does not address learning. Their main results show that summation of circuits that are both decomposable and either deterministic or based on an idempotent sum takes time linear in their size, whereas I show that decomposability alone is sufficient, a much weaker condition. In fact, over the same set of variables, deterministic circuits may be exponentially larger and are never smaller than non-deterministic circuits [31, 72]. Note that while decomposable circuits can be made deterministic by introducing hidden variables, this does not imply that these properties are equivalent.

Even when restricted to specific semirings, such as those for logical and probabilistic inference (e.g., Darwiche [33, 35], Poon and Domingos [114]), some of the following corollaries have not previously been shown formally, although some have been foreshadowed informally. Further, existing semiring-specific results (discussed in later sections) do not make it clear that the semiring properties are all that is required for tractable high-treewidth inference. The results presented in this dissertation are thus simpler and more general. Further, the sum-product theorem provides the basis for novel general algorithms for inference in arbitrary SPFs (Section 3.3) and for learning tractable high-treewidth representations (i.e., decomposable SPFs) in any semiring (Section 3.4).

3.2.1 Complexity of Summation for Decomposable SPFs

Let $S(\mathbf{X})$ be a decomposable SPF with size $|S|$ on commutative semiring $(R, \oplus, \otimes, 0, 1)$, let $d = |\mathcal{X}_i|$ for all $X_i \in \mathbf{X}$ where $\mathbf{X} = (X_1, \ldots, X_n)$, and let the cost of $a \oplus b$ and $a \otimes b$ for any elements $a, b \in R$ be c. Further, let e denote the complexity of evaluating any unary leaf function $\phi_j(X_i)$ in S and let $k = \max_{v \in S_{\text{sum}}, j \in \text{Ch}(v)} |\mathbf{X}_v \backslash \mathbf{X}_j| < n$, where S_{sum}, S_{prod}, and S_{leaf} are the sum, product, and leaf nodes in S, respectively, and $\text{Ch}(v)$ are the children of v. Then the complexity of computing $\bigoplus_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x})$ is $|S|c + |S_{\text{leaf}}|d(e + c) + |S_{\text{sum}}|(c + kdc)$.

For certain simple SPFs that have very little internal structure and many input variables, the worst case complexity of summing S can be quadratic in $|S|$ and occurs in the rare and restrictive case where $k = O(n) = O(|S|)$, due to the $\bigoplus_{\mathcal{X}_{v \backslash i}} 1$ term at each sum node (see

proof of Theorem 3.1). However, in any semiring with an idempotent sum (i.e., $a \oplus a = a$ for every $a \in R$) such as the min-sum or max-product semirings, this term is always equal to 1 and thus no computation is necessary. Alternatively, if the semiring supports multiplication and division, as the sum-product semiring does, then this complexity can be reduced by first computing the product over all variables and then dividing out as needed. If the semiring has neither of these properties, then the identity summations can still be computed with a single preprocessing pass through the SPF since they are constants and independent of the input variables. For all semirings studied in this dissertation, this quadratic cost does not occur, but I include it for completeness.

3.3 Inference in Non-Decomposable SPFs

Inference in arbitrary SPFs can be performed in a variety of ways, some more efficient than others. I present an algorithm for summing an SPF that adapts to the structure of the SPF and can thus take exponentially less time than constructing and summing a compatible tree-like SPF [8], which instead imposes a uniform decomposition structure. SPF S with root node r is summed by calling SumSPF(r), for which pseudocode is shown in Algorithm 1. SumSPF is a simple recursive algorithm for summing an SPF (note the similarity between its structure and the proof of the sum-product theorem). If S is decomposable, then SumSPF simply recurses to the bottom of S, sums the leaf functions, and evaluates S in an upward pass. If S is not decomposable, SumSPF decomposes each product node it encounters while summing S.

Decomposition can be achieved in many different ways, but I propose here a method that is based on a common algorithmic pattern that already occurs in many of the inference problems I consider, resulting in a general semiring-independent algorithm for summing any SPF. Decompose, shown in Algorithm 2, chooses a variable X_t that appears in the scope of multiple of v's children; creates $|\mathcal{X}_t|$ partially assigned and simplified copies S_{v_i} of the sub-SPF S_v for X_t assigned to each value $x_i \in \mathcal{X}_t$; multiplies each S_{v_i} by an indicator to ensure that only one is ever non-zero when S is evaluated; and then replaces v with a sum over $\{S_{v_i}\}$. Any node $u \in S_v$ that does not have X_t in its scope is re-used across each S_{v_i},

Algorithm 1 Sum an SPF.

Input: node v, the root of the sub-SPF $S_v(\mathbf{X}_v)$
Output: *sum*, which is equal to $\bigoplus_{\mathbf{v} \in \mathcal{X}_v} S_v(\mathbf{v})$

1: **function** SumSPF(v)
2: **if** $\langle v, sum \rangle$ in cache **then return** *sum*
3: **if** v is a sum node **then** // $\mathbf{X}_{v \setminus c} = \mathbf{X}_v \setminus \mathbf{X}_c$
4: $sum \leftarrow \bigoplus_{c \in \text{Ch}(v)} \text{SumSPF}(c) \otimes \bigoplus_{\mathcal{X}_{v \setminus c}} 1$
5: **else if** v is a product node **then**
6: **if** v is decomposable **then**
7: $sum \leftarrow \bigotimes_{c \in \text{Ch}(v)} \text{SumSPF}(c)$
8: **else**
9: $sum \leftarrow \text{SumSPF}(\text{Decompose}(v))$
10: **else** // *v is a leaf with constant a or function ϕ_v*
11: **if** v is a constant with value a **then**
12: $sum \leftarrow a$
13: **else**
14: $sum \leftarrow \bigoplus_{x_j \in \mathcal{X}_j} \phi_v(x_j)$
15: cache $\langle v, sum \rangle$
16: **return** *sum*

which can drastically limit the amount of duplication that occurs. Furthermore, each S_{v_i} is simplified by removing any nodes that became 0 when setting $X_t = x_i$. Variables are chosen heuristically; a good heuristic minimizes the amount of duplication that occurs. Similarly, SumSPF heuristically orders the children in lines 4 and 7. A good ordering will first evaluate children that may return an absorbing value (e.g., 0 for $\otimes$) because SumSPF can break out of these lines if this occurs.

In general, decomposing an SPF is hard, and the resulting decomposed SPF may be exponentially larger than the input SPF, although good heuristics can often avoid this. Many extensions to SumSPF are also possible, some of which I detail in later sections. In particular, the nonconvex optimization algorithm RDIS that I present in Chapter 4 is an extension of SumSPF for nonconvex optimization. Details about SumSPF for nonconvex optimization are presented in Section 3.5.4. Understanding inference in non-decomposable

Algorithm 2 Decompose a product node.

Input: product node v, with children $\{c\}$

Output: node s, such that its children are decomposable with respect to X_t and S_s, S_v are compatible

function DECOMPOSE(v)
  $X_t \leftarrow$ choose var. that appears in multiple $\mathbf{X}_c$
  $\mathbf{X}_{v \setminus t} \leftarrow \mathbf{X}_v \setminus \{X_t\}$
  $s \leftarrow$ create new sum node
  for all $x_i \in \mathcal{X}_t$ **do**
    create simplified $S_{v_i}(\mathbf{X}_{v \setminus t}) \leftarrow S_v(\mathbf{X}_{v \setminus t}, x_i)$
    set v_i as child of s // *v_i is the root of S_{v_i}*
    set $f(X_t) = [X_t = x_i]$ as child of v_i
  set s as a child of each of v's parents
  remove v and all edges containing v
  return s

SPFs is important for future work on extending SPF learning to even more challenging classes of functions, particularly those without obvious decomposability structure.

3.4 Learning Tractable Representations

Instead of performing inference in an intractable model, it can often be simpler to learn a tractable representation directly from data (e.g., Bach and Jordan [9], Gens and Domingos [48]). The general problem I consider here is that of learning a decomposable SPF $S : \mathcal{X} \to R$ on a semiring $(R, \oplus, \otimes, 0, 1)$ from a set of i.i.d. instances $T = \{(\mathbf{x}^{(i)}, y^{(i)})\}$ drawn from a fixed distribution $D_{\mathcal{X} \times R}$, where $y^{(i)} = \bigoplus_{\mathcal{Z}} F(\mathbf{x}^{(i)}, \mathbf{Z})$, F is some (unknown) SPF, and $\mathbf{Z}$ is a (possibly empty) set of unobserved variables or parameters with domain $\mathcal{Z}$, such that $S(\mathbf{x}^{(i)}) \approx y^{(i)}$, for all i. After learning, $\bigoplus_{\mathcal{X}} S(\mathbf{X})$ can be computed efficiently. In the sum-product semiring, this corresponds to summation (or integration), for which estimation of a joint probability distribution over $\mathbf{X}$ is a special case.

For certain problems, such as constraint satisfaction or MPE inference, the desired quantity is the argument of the sum. This can be recovered (if meaningful in the current

semiring) from an SPF by a single downward pass that recursively selects all children of a product node and the (or a) active child of a sum node (e.g., the child with the smallest value if minimizing). Learning for this domain corresponds to a generalization of learning for structured prediction [139]. Formally, the problem is to learn an SPF S from instances $T = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, where $\mathbf{y}^{(i)} = \arg\oplus_{\mathbf{y}\in\mathcal{Y}} F(\mathbf{x}^{(i)}, \mathbf{y})$, such that $\arg\oplus_{\mathbf{y}\in\mathcal{Y}} S(\mathbf{x}^{(i)}, \mathbf{y}) \approx \mathbf{y}^{(i)}$, for all i. Here, $\mathbf{x}^{(i)}$ can be an arbitrarily structured input and inference is over the variables $\mathbf{Y}$. Both of the above learning problems can be solved by the algorithm schema I present, with minor differences in the subroutines. I focus here on the former but discuss the latter below, alongside experiments on learning nonconvex functions that, by construction, can be efficiently optimized.

As shown by the sum-product theorem, the key to tractable inference is to identify the decomposability structure of an SPF. The difficulty, however, is that in general this structure varies throughout the space. For example, as a protein folds there exist conformations of the protein in which two particular amino acids are energetically independent (decomposable), and other conformations in which these amino acids directly interact, but in which other amino acids may no longer interact. This suggests a simple algorithm, which I call LearnSPF (shown in Algorithm 3), that first tries to identify a decomposable partition of the variables and, if successful, recurses on each subset of variables in order to find finer-grained decomposability. Otherwise, LearnSPF clusters the training instances, grouping those with analogous decomposition structure, and recurses on each cluster. Once either the set of variables is small enough to be summed over (in practice, unary leaf nodes are rarely necessary) or the number of instances is too small to contain meaningful statistical information, LearnSPF simply estimates an SPF $S(\mathbf{X})$ such that $S(\mathbf{x}^{(i)}) \approx y^{(i)}$ for all i in the current set of instances. LearnSPF is a generalization of LearnSPN [48], a simple but effective SPN structure learning algorithm.

LearnSPF is actually an algorithm schema that can be instantiated with different variable partitioning, clustering, and leaf creation subroutines for different semirings and problems. To successfully decompose the variables, LearnSPF must find a partition

Algorithm 3 Learn a decomposable SPF from data.

Input: a dataset $T = \{(\mathbf{x}^{(i)}, y^{(i)})\}$ over variables $\mathbf{X}$
Input: integer thresholds $t, v > 0$
Output: $S(\mathbf{X})$, an SPF over the input variables $\mathbf{X}$

function LEARNSPF($T, \mathbf{X}$)
  if $|T| \leq t$ or $|\mathbf{X}| \leq v$ **then**
    estimate $S(\mathbf{X})$ such that $S(\mathbf{x}^{(i)}) \approx y^{(i)}$ for all i
  else
    decompose $\mathbf{X}$ into disjoint subsets $\{\mathbf{X}_i\}$
    if $|\{\mathbf{X}_i\}| > 1$ **then**
      $S(\mathbf{X}) \leftarrow \bigotimes_i$ LEARNSPF($T, \mathbf{X}_i$)
    else
      cluster T into subsets of similar instances $\{T_j\}$
      $S(\mathbf{X}) \leftarrow \bigoplus_j$ LEARNSPF($T_j, \mathbf{X}$)
  return $S(\mathbf{X})$

$\{\mathbf{X}_1, \mathbf{X}_2\}$ of the variables $\mathbf{X}$ such that $\bigoplus_{\mathcal{X}} S(\mathbf{X}) \approx (\bigoplus_{\mathcal{X}_1} S_1(\mathbf{X}_1)) \otimes (\bigoplus_{\mathcal{X}_2} S_2(\mathbf{X}_2))$. I refer to this as *approximate* decomposability and discuss it further in Section 4.2.1. In probabilistic inference, mutual information or pairwise independence tests can be used to determine decomposability [48]. For my experiments, decomposable partitions correspond to the connected components of a graph over the variables in which correlated variables are connected. Instances can be clustered by virtually any clustering algorithm, such as a naive Bayes mixture model or k-means, which I use in my experiments. Instances can also be split by conditioning on specific values of the variables, as in SUMSPF or in a decision tree. Similarly, leaf functions can be estimated using any appropriate learning algorithm, such as linear regression or kernel density estimation.

In Section 3.6, I present preliminary experiments on learning nonconvex functions that can be globally optimized in polynomial time. However, this is just one particular application of LEARNSPF, which can be used to learn a tractable representation for any problem that consists of summation over a semiring. In the following section, I briefly discuss common inference problems that correspond to summing an SPF on a specific

semiring. For each, I demonstrate the benefit of the sum-product theorem, relate its core algorithms to SumSPF, and specify the problem solved by LearnSPF. Table 3.1 provides a summary of some of the relevant inference problems.

3.5 Applications to Specific Semirings

3.5.1 Logical Inference

Consider the Boolean semiring $\mathcal{B} = (\mathbb{B}, \vee, \wedge, 0, 1)$, where $\mathbb{B} = \{0, 1\}$, $\vee$ is logical disjunction (OR), and $\wedge$ is logical conjunction (AND). If each variable is Boolean and leaf functions are literals (i.e., each $\phi_l(X_j)$ is X_j or $\neg X_j$, where $\neg$ is logical negation), then SPFs on $\mathcal{B}$ correspond exactly to negation normal form (NNF), a DAG-based representation of a propositional formula (sentence) [11]. An NNF can be exponentially smaller than the same sentence in a standard (flat) representation such as conjunctive or disjunctive normal form (CNF or DNF), and is never larger [31]. Summation of an NNF $F(\mathbf{X})$ on $\mathcal{B}$ is $\bigvee_{\mathcal{X}} F(\mathbf{X})$, which corresponds to propositional satisfiability (SAT): the problem of determining if there exists a satisfying assignment for F. Thus, the tractability of SAT for decomposable NNFs follows directly from the sum-product theorem.

Corollary 3.3 (Darwiche [33])**.** *The satisfiability of a decomposable NNF is decidable in time linear in its size.*

Satisfiability of an arbitrary NNF can be determined either by decomposing the NNF or by expanding it to a CNF and using a SAT solver. DPLL [37], the standard algorithm for solving SAT, is an instance of SumSPF (see also Huang and Darwiche [63]). Specifically, DPLL is a recursive algorithm that at each level chooses a variable $X \in \mathbf{X}$ for CNF $F(\mathbf{X})$ and computes $F = F|_{X=0} \vee F|_{X=1}$ by recursing on each disjunct, where $F|_{X=x}$ is F with X assigned value x. Thus, each level of recursion of DPLL corresponds to a call to Decompose.

Learning in the Boolean semiring is a well-studied area that includes problems from learning Boolean circuits [67] (of which decomposable SPFs are a restricted subclass, known as syntactically multilinear circuits) to learning sets of rules [117]. However, learned rule

Domain	Inference task	Semiring	Variables	Leaf functions	SumSPF
Logical inference	SAT	$(\mathbb{B}, \vee, \wedge, 0, 1)$	Boolean	Literals	DPLL
	#SAT	$(\mathbb{N}, +, \times, 0, 1)$	Boolean	Literals	#DPLL
	MAX-SAT	$(\mathbb{N}_{-\infty}, \max, +, -\infty, 0)$	Boolean	Literals	MPE-SAT
Constraint satisfaction	CSPs	$(\mathbb{B}, \vee, \wedge, 0, 1)$	Discrete	Univariate constraints	Backtracking
	Fuzzy CSPs	$([0, 1], \max, \min, 0, 1)$	Discrete	Univariate constraints	-
	Weighted CSPs	$(\mathbb{R}_{+,\infty}, \min, +, \infty, 0)$	Discrete	Univariate constraints	-
Probabilistic inference	Marginal	$(\mathbb{R}_{+}, +, \times, 0, 1)$	Discrete	Potentials	Recursive conditioning
	MPE	$(\mathbb{R}_{+}, \max, \times, 0, 1)$	Discrete	Potentials	
Continuous functions	Integration	$(\mathbb{R}_{+}, +, \times, 0, 1)$	Continuous	Univariate functions	-
	Optimization	$(\mathbb{R}_{\infty}, \min, +, \infty, 0)$	Continuous	Univariate functions	RDIS
Relational databases	Unions of CQs	$(\mathbf{U}_m, \cup, \bowtie, \varnothing, \mathbf{1}_R)$	Sets of tuples	Unary tuples	Generic-Join
	Provenance	$(\mathbb{N}[\mathbf{X}], +, \times, 0, 1)$	Discrete	K-relation tuples	-

Table 3.1: A subset of the inference problems that correspond to summing an SPF on a specific semiring, with details on the variables and leaf functions and a core algorithm that is an instance of SumSPF. $\mathbb{B} = \{0, 1\}$. $\mathbb{N}$ and $\mathbb{R}$ denote the natural and real numbers. Subscript $+$ denotes the restriction to non-negative numbers and subscript $(-)\infty$ denotes the inclusion of (negative) ∞. $\mathbf{U}_m$ denotes the universe of relations of arity up to m (see Section 3.5.5). $\mathbb{N}[\mathbf{X}]$ denotes the polynomials with coefficients from $\mathbb{N}$. See sections 3.5.1 and 3.5.5 for information on MPE-SAT [128] and Generic-Join [107], respectively. RDIS is presented in Chapter 4.

sets are typically encoded in large CNF knowledge bases, making reasoning over them intractable. In contrast, decomposable NNF is a tractable but expressive formalism for knowledge representation that supports a rich class of polynomial-time logical operations, including SAT [33]. Thus, LearnSPF in this semiring provides a method for learning large, complex knowledge bases that are encoded in decomposable NNF and therefore support efficient querying, which could greatly benefit existing rule learning systems.

Model Counting

Model counting (#SAT) is the problem of computing the number of satisfying assignments of a Boolean formula. The model count of an NNF F can be obtained by *translating* it from the Boolean semiring to the counting sum-product semiring $\mathcal{P} = (\mathbb{N}, +, \times, 0, 1)$ ($\mathbb{R}_+$ is used instead for weighted #SAT), and then summing it.

Definition 3.7. Translating *an SPF from semiring* $(R, \oplus, \otimes, 0, 1)$ *to semiring* $(R', \boxplus, \boxtimes, 0', 1')$ *with* $R \subseteq R'$*, involves replacing each* $\oplus$ *node with a* $\boxplus$ *node, each* $\otimes$ *node with a* $\boxtimes$ *node, and each leaf function that returns* 0 *or* 1 *with one that returns* $0'$ *or* $1'$*, respectively.*

However, simply summing the translated function F' may compute an incorrect model count because the same satisfying assignment may be counted multiple times; this occurs when the idempotence (a semiring R is idempotent if $a \oplus a = a$ for $a \in R$) of the two semirings differs, i.e., when semiring R is idempotent and R' is not, and vice versa. If exactly one of the two semirings is idempotent, F must be *deterministic* to ensure that summing F' gives the correct model count.

Definition 3.8. *An OR node is* deterministic *iff the supports of its children are disjoint. An NNF is deterministic iff all of its OR nodes are deterministic.*

The support of a function $G(\mathbf{X})$ is the set of points $\mathcal{S} \subseteq \mathcal{X}$ such that $G(\mathbf{x}) \neq 0$ for all $\mathbf{x} \in \mathcal{S}$. If F is deterministic and decomposable, then it follows from the sum-product theorem that its model count can be computed efficiently.

Corollary 3.4 (Darwiche [32])**.** *The model count of a deterministic, decomposable NNF can be computed in time linear in its size.*

Proof. Let $F(\mathbf{X})$ be a deterministic, decomposable NNF and $F'(\mathbf{X})$ be F translated to the sum-product semiring. Clearly, F and F' have equal size and F' is deterministic and decomposable. Thus, from the sum-product theorem, $\sum_{\mathcal{X}} F'(\mathbf{X})$ takes time linear in the size of F. Let v be a node in F and v' its corresponding node in F'. It remains to show that $\sum_{\mathcal{X}} F'_v(\mathbf{X}) = \#\text{SAT}(F_v(\mathbf{X}))$ for all v, v', which I do by induction. The base case with v a leaf node holds trivially. For the induction step, assume that $\sum_{\mathcal{X}_i} F'_i(\mathbf{X}_i) = \#\text{SAT}(F_i(\mathbf{X}_i))$ for each child $c_i \in \text{Ch}(v)$ (resp. $c'_i \in \text{Ch}(v')$). If v is an AND node then v' is a multiplication node and $\#\text{SAT}(F_v(\mathbf{X})) = \#\text{SAT}(\bigwedge_{c_i} F_i(\mathbf{x}_i)) = \prod_{c_i} \#\text{SAT}(F_i(\mathbf{x}_i)) = \sum_{\mathcal{X}} F'_v(\mathbf{X})$, because v and v' are decomposable. If v is an OR node then v' is an addition node and $\#\text{SAT}(F_v(\mathbf{X})) = \#\text{SAT}(\bigvee_{c_i} F_i(\mathbf{x}_i)) = \sum_{c_i} \#\text{SAT}(F_i(\mathbf{x}_i)) = \sum_{\mathcal{X}} F'_v(\mathbf{X})$, because v is deterministic, so its children are logically disjoint. □

Most algorithms for #SAT (e.g., Relsat [12]), Cachet [126], #DPLL [8]) are also instances of SumSPF, since they extend DPLL by, at each level of recursion, decomposing the CNF into independent components (i.e., no variable appears in multiple components), solving these separately, and caching the model count of each component. Component decomposition corresponds to a decomposable product node in SumSPF and component caching corresponds to connecting a sub-SPF to multiple parents. Notice that the sum nodes created by Decompose are deterministic.

The Maximum Satisfiability Problem (MAX-SAT)

MAX-SAT is the problem of computing the maximum number of satisfiable clauses of a CNF, over all assignments. It can be generalized to NNFs as follows.

Definition 3.9. *Let $F(\mathbf{X})$ be an NNF and $\mathbf{x} \in \mathcal{X}$ an assignment. The* SAT number (*SN*) *of a literal $\phi(X_j) \in F$ is 1 if $\phi(\mathbf{x}_j)$ is true and 0 otherwise. The SN of an AND node is the sum of the SNs of its children. The SN of an OR node is the max of the SNs of its children.*

MAX-SAT of an NNF $F(\mathbf{X})$ is the problem of computing the maximum SAT number of the root of F over all assignments $\mathbf{x} \in \mathcal{X}$. If F is a CNF, then this reduces to standard MAX-SAT. MAX-SAT of F can be solved by translating F to the max-sum semiring $\mathcal{M} = (\mathbb{N}_{-\infty}, \max, +, -\infty, 0)$ (where $\mathbb{R}_{+,-\infty}$ is used for weighted MAX-SAT), and then summing it. Clearly, F' is an SPF on $\mathcal{M}$, i.e., a max-sum network. The corollary below follows immediately from the sum-product theorem.

Corollary 3.5 (Darwiche [33])**.** *MAX-SAT of a decomposable NNF can be computed in time linear in its size.*

MAX-SAT of an arbitrary NNF (or CNF) can be computed by first translating it to $\mathcal{M}$ and then calling SumSPF, which can be extended to perform branch and bound (BnB) [86] when traversing the SPF. This allows SumSPF to prune sub-SPFs that are not relevant to the final solution, which can greatly reduce the search space. With this addition, DPLL-based BnB solvers for MAX-SAT (e.g., Heras et al. [59] and references therein) are instances of SumSPF. Most relevant, however, is the MPE-SAT algorithm of Sang et al. [128], since both it and SumSPF use decomposition and caching to improve their efficiency.

3.5.2 Constraint Satisfaction

A constraint satisfaction problem (CSP) consists of a set of constraints $\{C_i\}$ on variables $\mathbf{X}$, where each constraint $C_i(\mathbf{X}_i)$ specifies the satisfying assignments to its variables $\mathbf{X}_i \subseteq \mathbf{X}$. Solving a CSP consists of finding an assignment to $\mathbf{X}$ that satisfies each constraint. When constraints are functions $C_i : \mathcal{X}_i \to \mathbb{B}$ that are 1 when C_i is satisfied and 0 otherwise, then

$$\begin{aligned} F(\mathbf{X}) &= \bigwedge_i C_i(\mathbf{X}_i) \\ &= \bigwedge_i \bigvee_{\mathbf{x}_i \in \mathcal{X}_i} (C_i(\mathbf{x}_i) \wedge [\mathbf{X}_i = \mathbf{x}_i]) \\ &= \bigwedge_i \bigvee_{\mathbf{x}_i \in \mathcal{X}_i} \left(C_i(\mathbf{x}_i) \wedge \bigwedge_{X_t \in \mathbf{X}_i} [X_t = \mathbf{x}_{it}] \right) \end{aligned}$$

is a CSP and F is an SPF on the Boolean semiring $\mathcal{B}$, i.e., an OR-AND network (OAN), a generalization of NNF, and a decomposable CSP is one with a decomposable OAN. Solving F corresponds to computing $\bigvee_{\mathcal{X}} F(\mathbf{X})$, which is summation on $\mathcal{B}$ (see also Bistarelli et al. [17], Chang and Mackworth [26], Rollon et al. [118]). The solution for F can be recovered with a downward pass that recursively selects the (or a) non-zero child of an OR node, and all children of an AND node. Corollary 3.6 follows immediately.

Corollary 3.6. *Every decomposable CSP can be solved in time linear in its size.*

Thus, for inference to be efficient it suffices that the CSP be expressible by a tractably-sized decomposable OAN; a much weaker condition than that of low treewidth. Like DPLL, backtracking-based search algorithms [83] for CSPs are also instances of SumSPF (see also Mateescu and Dechter [100]). Further, SPFs on a number of other semirings correspond to various extensions of CSPs, including fuzzy, probabilistic, and weighted CSPs (see Table 3.1 and Bistarelli et al. [17]).

LearnSPF for CSPs addresses a variant of structured prediction [139]; specifically, learning a function $F : \mathbf{X} \to \mathbb{B}$ such that $\arg \bigvee_{\mathbf{y}} F(\mathbf{x}^{(i)}, \mathbf{y}) \approx \mathbf{y}^{(i)}$ for training data $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, where $\mathbf{x}^{(i)}$ is a structured object representing a CSP and $\mathbf{y}^{(i)}$ is its solution. LearnSPF solves this problem while guaranteeing that the learned CSP remains tractable. This is a much simpler and more attractive approach than existing constraint learning methods such as Lallouet et al. [84], which uses inductive logic programming and has no tractability guarantees.

3.5.3 Probabilistic Inference

Recall that many probability distributions can be compactly represented as graphical models: $P(\mathbf{X}) = \frac{1}{Z} \prod_i \psi_i(\mathbf{X}_i)$, where ψ_i is a potential over variables $\mathbf{X}_i \subseteq \mathbf{X}$ and Z is known as the partition function [111]. One of the main inference problems in graphical models is to compute the probability of evidence $\mathbf{e} \in \mathcal{X}_E$ for variables $\mathbf{X}_E \subseteq \mathbf{X}$, $P(\mathbf{e}) = \sum_{\mathcal{X}_{\overline{E}}} P(\mathbf{e}, \mathbf{X}_{\overline{E}})$, where $\mathbf{X}_{\overline{E}} = \mathbf{X} \backslash \mathbf{X}_E$. The partition function Z is the unnormalized probability of empty

evidence ($\mathbf{X}_E = \varnothing$). Unfortunately, computing Z or $P(\mathbf{e})$ is generally intractable. Building on a number of earlier works [8, 35, 40], Poon and Domingos [114] introduced sum-product networks (SPNs), a class of distributions in which inference is guaranteed to be tractable. An SPN is an SPF on the non-negative real sum-product semiring $(\mathbb{R}_+, +, \times, 0, 1)$. A graphical model is a flat SPN, in the same way that a CNF is a flat NNF [31]. For an SPN S, the unnormalized probability of evidence $\mathbf{e} \in \mathcal{X}_E$ for variables $\mathbf{X}_E$ is computed by replacing each leaf function $\phi_l \in \{\phi_l(X_j) \in S : X_j \in \mathbf{X}_E\}$ with the constant $\phi_l(\mathbf{e}_j)$ and summing the SPN. The corollary below follows immediately from the sum-product theorem.

Corollary 3.7. *The probability of evidence in a decomposable SPN can be computed in time linear in its size.*

A similar result (shown below) for finding the most probable state of the non-evidence variables also follows from the sum-product theorem. One important consequence of the sum-product theorem is that decomposability is the sole condition required for an SPN to be tractable; previously, completeness was also required [48, 114]. This expands the range of tractable SPNs and simplifies the design of tractable representations based on them, such as tractable probabilistic knowledge bases [42] and submodular field grammars (Chapter 5).

Most existing algorithms for inference in graphical models correspond to different methods of decomposing a flat SPN, and can be loosely clustered into tree-based, conditioning, and compilation methods, all of which SumSPF generalizes. Tree-based methods include junction-tree clustering [85] and variable elimination [39], which correspond (explicitly and implicitly, respectively) to constructing a junction tree and then summing its corresponding tree-like SPN. Conditioning algorithms such as recursive conditioning [34], value elimination [7], AND/OR search [40], and #DPLL [8] traverse the space of partial assignments by recursively conditioning on variables and their values. These algorithms vary in the flexibility of their variable ordering, decomposition, and caching (see Bacchus et al. [8] for a comparison), but are all instances of SumSPF, which can use a fully-dynamic

variable ordering, as value elimination can and #DPLL does, or a fixed ordering, as in variants of recursive conditioning and AND/OR search. Decomposition and caching correspond to decomposable product nodes and connecting sub-SPNs to multiple parents, respectively, in SumSPF. Thirdly, inference in graphical models can be performed by compilation to an arithmetic circuit (AC) [35]. In discrete domains, Rooshenas and Lowd [119] showed that SPNs and ACs are equivalent, but that SPNs are always smaller or equal in size. In continuous domains, however, it is unlikely that even this relationship exists, because an AC would require an infinite number of indicator functions. Furthermore, existing compilation methods require first encoding the graphical model in very restrictive languages (such as CNF or SDDs), which can make them exponentially slower than SumSPF. Finally, no tractability properties have been established for ACs so there is no guarantee before compiling that inference will be tractable.

LearnSPF for SPNs corresponds to learning a probability distribution from a set of samples $\{(\mathbf{x}^{(i)}, y^{(i)})\}$. Note that $y^{(i)}$ in this case is defined implicitly by the empirical frequency of $\mathbf{x}^{(i)}$ in the dataset. Learning the parameters and structure of SPNs is a fast-growing area of research (e.g., Adel et al. [2], Gens and Domingos [48], Peharz et al. [112], Rooshenas and Lowd [119]), and I refer readers to these references for more details.

Most-Probable Explanation (MPE)

Beyond computing the probability of evidence, another important probabilistic inference problem is finding the most probable or MPE state of the non-evidence variables of $P(\mathbf{X})$ given the evidence: $\arg\max_{\mathcal{X}_{\overline{E}}} P(\mathbf{e}, \mathbf{X}_{\overline{E}})$ for evidence $\mathbf{e} \in \mathcal{X}_E$ where $\mathbf{X}_{\overline{E}} = \mathbf{X} \backslash \mathbf{X}_E$. The MPE value (maximum probability of any state) of an SPN S can be computed by translating S to the non-negative max-product semiring $(\mathbb{R}_+, \max, \times, 0, 1)$ and maximizing the resulting SPF S'. The MPE state can then be recovered by a downward pass in S', recursively selecting the (or a) highest-valued child of each max node and all children of each product node [114]. As when translating an NNF for model counting, an SPN must be *selective* [112] (the SPN equivalent of deterministic) for summation in the max-product semiring to give the correct MPE.

Corollary 3.8. *The MPE state of a selective, decomposable SPN can be found in time linear in its size.*

Proof. Let $S(\mathbf{X})$ be a selective, decomposable SPN and $S'(\mathbf{X})$ be its max-product version, which has the same size and is also selective and decomposable. For clarity, I assume that there is no evidence, as it is trivial to incorporate. From the sum-product theorem, $\max_{\mathcal{X}} S'(\mathbf{X})$ takes time linear in the size of S. Let v be a node in S and v' its corresponding node in S'. It remains to show that $\max_{\mathcal{X}} S'_v(\mathbf{X}) = \max_{\mathcal{X}} S(\mathbf{X})$ for all v, v', which I do by induction on v. The base case with v a leaf holds trivially because v and v' are identical. For the induction step, assume that $\max_{\mathcal{X}_i} S'_i(\mathbf{X}_i) = \max_{\mathcal{X}_i} S_i(\mathbf{X}_i)$ for each child $c_i \in \text{Ch}(v)$ (resp. $c'_i \in \text{Ch}(v')$). If v is a product node then so is v' and $\max_{\mathcal{X}} S'_v(\mathbf{X}) = \max_{\mathcal{X}} S_v(\mathbf{X})$. If v is a sum node then v' is a max node and

$$\begin{aligned}
\max_{\mathcal{X}} S(\mathbf{X}) &= \max_{\mathbf{x}\in\mathcal{X}} \sum_{c_i} S_i(\mathbf{x}_i) \\
&= \max_{\mathbf{x}\in\mathcal{X}} \left\{ \max_{c_i} S_i(\mathbf{x}_i) \right\} \\
&= \max_{c_i} \left\{ \max_{\mathbf{x}_i\in\mathcal{X}_i} S_i(\mathbf{x}_i) \right\} \\
&= \max_{\mathcal{X}} S'(\mathbf{X}),
\end{aligned}$$

where the second equality occurs because v is selective. After summing S', the MPE state is recovered by a downward pass in S', which takes linear time. □

A sum node in an SPN can be viewed as the result of summing out an implicit hidden variable Y_v, whose values $\mathcal{Y}_v = \{y_c\}_{c\in\text{Ch}(v)}$ correspond to $\text{Ch}(v)$, the children of v [114]. It is often of interest to find the MPE state of both the hidden and observed variables. This can be done in linear time and requires only that the SPN be decomposable, because making each Y_v explicit by multiplying each child c of v by the indicator $[Y_v = y_c]$ makes the resulting SPN $S(\mathbf{X}, \mathbf{Y})$ selective.

3.5.4 Integration and Optimization

SPFs can be generalized to continuous (real) domains, where each variable X_i has domain $\mathcal{X}_i \subseteq \mathbb{R}$ and the semiring set is a subset of $\mathbb{R}_\infty$. For the sum-product theorem to hold, the only additional conditions are that (C1) $\bigoplus_{X_j} \phi_l(X_j)$ is computable in constant time for all leaf functions, and (C2) $\bigoplus_{\mathcal{X}_{v\setminus c}} 1 \neq \infty$ for all sum nodes $v \in S$ and all children $c \in \mathrm{Ch}(v)$, where $\mathcal{X}_{v\setminus c}$ is the domain of $\mathbf{X}_{v\setminus c} = \mathbf{X}_v \setminus \mathbf{X}_c$.

Integration

In the non-negative real sum-product semiring $(\mathbb{R}_+, +, \times, 0, 1)$, summation of an SPF with continuous variables corresponds to integration over $\mathcal{X}$. Accordingly, I generalize SPFs as follows. Let $\mu_1, \ldots, \mu_n$ be measures over $\mathcal{X}_1, \ldots, \mathcal{X}_n$, respectively, where each leaf function $\phi_l : \mathcal{X}_j \to \mathbb{R}_+$ is integrable with respect to μ_j, which satisfies (C1). Summation (integration) of an SPF $S(\mathbf{X})$ then corresponds to computing $\int_{\mathcal{X}} S(\mathbf{X}) d\mu = \int_{\mathcal{X}_1} \cdots \int_{\mathcal{X}_n} S(\mathbf{X}) d\mu_1 \cdots d\mu_n$. For (C2), $\bigoplus_{\mathcal{X}_{v\setminus c}} 1 = \int_{\mathcal{X}_{v\setminus c}} 1 \, d\mu_{v\setminus c}$ must be integrable for all sum nodes $v \in S$ and all children $c \in \mathrm{Ch}(v)$, where $d\mu_{v\setminus c} = \prod_{\{j : X_j \in \mathbf{X}_{v\setminus c}\}} d\mu_j$. I thus assume that either $\mu_{v\setminus c}$ has finite support over $\mathcal{X}_{v\setminus c}$ or that $\mathbf{X}_{v\setminus c} = \varnothing$. Corollary 3.9 follows immediately.

Corollary 3.9. *Every decomposable SPF of real variables can be integrated in time linear in its size.*

Thus, decomposable SPFs define a class of functions for which exact integration is tractable. SumSPF defines a novel algorithm for (approximate) integration that is based on recursive problem decomposition, and can be exponentially more efficient than standard integration algorithms such as trapezoidal or Monte Carlo methods [115] because it dynamically decomposes the problem at each recursion level and caches intermediate computations. For non-decomposable SPFs, Decompose must be altered to select only a finite number of values and then use the trapezoidal rule for approximate integration. Values can be chosen using grid search and if S is Lipschitz continuous the grid spacing can be set such that the error incurred by the approximation is bounded by a pre-specified amount. This can significantly reduce the number of values explored in SumSPF

if combined with approximate decomposability (Section 3.4), since SumSPF can treat some non-decomposable product nodes as decomposable, avoiding the expensive call to Decompose while incurring only a bounded integration error.

In this semiring, LearnSPF learns a decomposable continuous SPF $S : \mathcal{X} \to \mathbb{R}_+$ on samples $\{(\mathbf{x}^{(i)}, y^{(i)} = F(\mathbf{x}^{(i)}))\}$ from an SPF $F : \mathcal{X} \to \mathbb{R}_+$, where S can be integrated efficiently over the domain $\mathcal{X}$. Thus, LearnSPF provides a novel method for learning complex functions that are easily integrable. This could be very beneficial if used to learn the partition function of a continuous probability distribution, for example.

Nonconvex optimization

Summing a continuous SPF in one of the min-sum, min-product, max-sum, or max-product semirings corresponds to optimizing a (potentially nonconvex) continuous objective function. My results hold for all of these, but I focus here on the real min-sum semiring $(\mathbb{R}_\infty, \min, +, \infty, 0)$, where summation of a min-sum function (MSF) $F(\mathbf{X})$ corresponds to computing $\min_{\mathcal{X}} F(\mathbf{X})$. A flat MSF is simply a sum of terms. To satisfy (C1), I assume that $\min_{x_j \in \mathcal{X}_j} \phi_l(x_j)$ is computable in constant time for all $\phi_l \in F$. (C2) is trivially satisfied for $\min$. The corollary below follows immediately.

Corollary 3.10. *The global minimum of a decomposable MSF can be found in time linear in its size.*

Corollary 3.10 defines a novel class of functions that can be efficiently globally optimized. SumSPF provides an outline for a general nonconvex optimization algorithm for sum-of-terms (or product-of-factors) functions. The RDIS algorithm for nonconvex optimization – presented in Chapter 4 and in Friesen and Domingos [44] – achieves exponential speedups compared to other algorithms and is an instance of SumSPF where values are chosen via multi-start gradient descent and variables in Decompose are chosen by graph partitioning.

For nonconvex optimization, LearnSPF solves a variant of structured prediction [139], in which the variables to predict are continuous instead of discrete (e.g., protein folding,

structure from motion [44]). The training data is a set $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, where $\mathbf{x}^{(i)}$ is a structured object representing a nonconvex function and $\mathbf{y}^{(i)}$ is a vector of values specifying the global minimum of that function. LEARNSPF learns a function $S : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_\infty$ such that $\arg\min_{\mathbf{y}\in\mathcal{Y}} S(\mathbf{x}^{(i)}, \mathbf{y}) \approx \mathbf{y}^{(i)}$, where the $\arg\min$ can be computed efficiently because S is decomposable. More detail is provided in Section 3.6, which presents experimental results on using LEARNSPF to learn an MSF.

3.5.5 Relational Inference

Let $\mathcal{X}$ be a finite set of constants and let $R^k = \mathcal{X}^k$ be the complete relation[1] of arity k on $\mathcal{X}$, i.e., the set of all tuples in $\mathcal{X}^k$, where $\mathcal{X}^k$ is the Cartesian product of $\mathcal{X}$ with itself $k-1$ times. The universe of relations with arity up to m is $\mathbf{U}_m = \mathbf{1}_R \cup \bigcup_{i=1}^{m} 2^{R^i}$, where 2^{R^k} is the power set of R^k and $\mathbf{1}_R$ is the (identity) relation containing the empty tuple. Since union distributes over join, and both are associative and commutative, $\mathcal{R} = (\mathbf{U}_m, \cup, \bowtie, \varnothing, \mathbf{1}_R)$ is a semiring over relations, where $\bowtie$ is natural join and $\varnothing$ is the empty set (recall that $R = R \bowtie \mathbf{1}_R$ for any relation R). Given an extensional database $\mathbf{R} = \{R_i\}$ containing relations R_i of arity up to m, an SPF on $\mathcal{R}$, referred to as a union-join network (UJN), is a query on $\mathbf{R}$. In a UJN Q, each $R_i \in \mathbf{R}$ is composed of a union of joins of unary tuples, such that $R_i = \bigcup_{\langle c_1,\ldots,c_r\rangle\in R_i} \bowtie_{j=1}^{r} c_j$, where the leaves of Q are the unary tuples c_j. The R_i are then combined with unions and joins to form the full UJN (query). A UJN $Q(\mathbf{X})$ over query variables $\mathbf{X} = (X_1, \ldots, X_n)$ defines an intensional output relation $Q_{\text{ans}} = \bigcup_{\mathcal{X}^n} Q(\mathbf{X})$. Clearly, computing Q_{ans} corresponds to summation in $\mathcal{R}$. Let n_J^Q denote the maximum number of variables involved in a particular join over all joins in Q. The corollary below follows immediately, since a decomposable join is a Cartesian product.

Corollary 3.11. *Q_{ans} of a decomposable UJN Q can be computed in time linear in the size of Q if n_J^Q is bounded.*

Note that n_J^Q can be smaller than the treewidth of Q, since Q composes the final output

[1] A relation is a set of tuples; see Abiteboul et al. [1] for details on relational databases.

relation from many small relations (starting with unary tuples) via a relational form of determinism. Since the size of a UJN depends both on the input relations and the query, this is a statement about the combined complexity of queries defined by UJNs.

Regarding expressivity, selection in a UJN can be implemented as a join with the relation $P_\sigma \in \mathbf{U}_m$, which contains all tuples that satisfy the selection predicate σ. Projection is not immediately supported by $\mathcal{R}$, but since union distributes over projection, it is straightforward to extend the results of Theorem 3.1 to allow UJNs to contain projection nodes. UJNs with projection correspond to non-recursive Datalog queries (i.e., unions of conjunctive queries), for which decomposable UJNs are a tractable subclass. Thus, SumSPF defines a recursive algorithm for evaluating non-recursive Datalog queries and the Generic-Join algorithm [107] – a recent join algorithm that achieves worst-case optimal performance by recursing on individual tuples – is an instance of Decompose.

3.5.6 Relational Probabilistic Models

Another benefit of the sum-product theorem is that it enables a much simpler proof of the tractability of tractable probabilistic knowledge bases (TPKB) [108, 146], a practical and efficient representation that combines first-order logic and probability theory and is based on tractable Markov logic [42]. I present this proof below.

A tractable probabilistic knowledge base (TPKB) is a set of class and object declarations such that the classes form a forest and the objects form a tree of subparts when given the leaf class of each object. A class declaration for a class $\mathtt{C}$ specifies the subparts $\mathtt{Parts}(\mathtt{C}) = \{\mathtt{P_i}\}$, (weighted) subclasses $\mathtt{Subs}(\mathtt{C}) = \{\mathtt{S_i}\}$, attributes $\mathtt{Atts}(\mathtt{C}) = \{\mathtt{A_i}\}$, and (weighted) relations $\mathtt{Rels}(\mathtt{C}) = \{\mathtt{R_i}\}$. The subparts of $\mathtt{C}$ are parts that every object of class $\mathtt{C}$ must have and are specified by a name $\mathtt{P_i}$, a class $\mathtt{C_i}$, and a number n_i of unique copies. A class $\mathtt{C}$ with subclasses $\mathtt{S}_1, \ldots, \mathtt{S_j}$ must belong to exactly one of these subclasses, where the weights w_i specify the distribution over subclasses. Every attribute has a domain $\mathbf{D_i}$ and a weight function $\mathbf{u_i} : \mathbf{D_i} \to \mathbb{R}$. Each relation $\mathtt{R_i}(\ldots)$ has the form $\mathtt{R_i}(\mathtt{P_a}, \ldots, \mathtt{P_z})$ where each of $\mathtt{P_a}, \ldots, \mathtt{P_z}$ is a part of $\mathtt{C}$. Relations specify what relationships may hold among the subparts.

A weight v_i on $\mathtt{R_i}$ defines the probability that the relation is true. A relation can also apply to the object as a whole, instead of to its parts. Object declarations introduce evidence by specifying an object's subclass memberships, attribute values, and relations as well as specifying the names and path of the object from the top object in the part decomposition.

A TPKB $\mathcal{K}$ is a DAG of objects and their properties (classes, attributes, and relations), and a possible world $\mathbf{W}$ is a subtree of the DAG with values for the attributes and relations. The literals are the class membership, attribute, and relation atoms and their negations and thus specify the subclasses of each object, the truth value of each relation, and the value of each attribute. A single (root) top object $(\mathtt{O_0}, \mathtt{C_0})$ has all other objects as descendants. No other objects are of top class $\mathtt{C_0}$. The unnormalized distribution ϕ over possible subworlds $\mathbf{W}$ is defined recursively as $\phi(\mathtt{O}, \mathtt{C}, \mathbf{W}) = 0$ if $\neg\mathtt{Is}(\mathtt{O}, \mathtt{C}) \in \mathbf{W}$ or if a relation $\mathtt{R}$ of $\mathtt{C}$ is hard and $\neg\mathtt{R}(\mathtt{O}, \dots) \in \mathbf{W}$, and otherwise as

$$\phi(\mathtt{O}, \mathtt{C}, \mathbf{W}) = \left(\sum_{\mathtt{S_i} \in \mathtt{Subs}(\mathtt{C})} e^{w_i} \phi(\mathtt{O}, \mathtt{S_i}, \mathbf{W}) \right) \times \left(\prod_{\mathtt{P_i} \in \mathtt{Parts}(\mathtt{C})} \phi(\mathtt{O}.\mathtt{P_i}, \mathtt{C_i}, \mathbf{W}) \right) \times \left(\prod_{\mathtt{A_i} \in \mathtt{Atts}(\mathtt{C})} \alpha(\mathtt{O}, \mathtt{A_i}, \mathbf{W}) \right) \times \left(\prod_{\mathtt{R_i} \in \mathtt{Rels}(\mathtt{C})} \rho(\mathtt{O}, \mathtt{R_i}, \mathbf{W}) \right), \tag{3.1}$$

where $\alpha(\mathtt{O}, \mathtt{A_i}, \mathbf{W}) = e^{\mathbf{u_i}(D)}$ if $\mathtt{A_i}(\mathtt{O}, \mathtt{D}) \in \mathbf{W}$ and $\rho(\mathtt{O}, \mathtt{R_i}, \mathbf{W}) = e^{v_i}[\mathtt{R_i}(\dots)] + [\neg\mathtt{R_i}(\dots)]$. Note that $\mathtt{Parts}(\mathtt{C})$ contains all subparts of $\mathtt{C}$, including all duplicated parts. The probability of a possible world $\mathbf{W}$ is $\frac{1}{Z_\mathcal{K}} \phi(\mathtt{O_0}, \mathtt{C_0}, \mathbf{W})$ where the sub-partition function for $(\mathtt{O}, \mathtt{C})$ is $Z_{\mathcal{K}_{\mathtt{O},\mathtt{C}}} = \sum_{\mathbf{W} \in \mathcal{W}} \phi(\mathtt{O}, \mathtt{C}, \mathbf{W})$ and $Z_\mathcal{K} = Z_{\mathcal{K}_{\mathtt{O_0},\mathtt{C_0}}}$.

By construction, $\phi(\mathtt{O_0}, \mathtt{C_0}, \mathbf{W})$ defines an SPN over the literals. With the sum-product theorem in hand, it is possible to greatly simplify the two-page proof of tractability given in Niepert and Domingos [108], as I show here. To prove that computing $Z_\mathcal{K}$ is tractable it suffices to show that equation (3.1) is decomposable or can be decomposed efficiently. I first note that each of the four factors in equation (3.1) is decomposable, since the first is a sum, the second is a product over the subparts of O and therefore its subfunctions have

disjoint scopes, and the third and fourth are products over the attributes and relations, respectively, and are decomposable because none of the $\alpha(\cdot)$ or $\rho(\cdot)$ share variables. It only remains to show that the factors can be decomposed with respect to each other without increasing the size of the SPN. Let n_O, n_C, n_r denote the number of object declarations, class declarations, and relation rules, respectively. The SPN corresponding to $\phi(\mathtt{O}_0, \mathtt{C}_0, \mathbf{W})$ has size $O(n_O(n_C+n_r))$, since for each object $(\mathtt{O}, \mathtt{C})$ there are a constant number of edges for each of its relations and subclasses. Similarly, $\mathcal{K}$ has size $|\mathcal{K}|= O(n_O(n_C + n_r)$. Corollary 3.12 can now be proved.

Corollary 3.12 (Niepert and Domingos [108])**.** *The partition function of a TPKB can be computed in time linear in its size.*

Proof. The only sources of non-decomposability in (3.1) are if an object $(\mathtt{O}, \mathtt{C})$ and one of its subclasses $(\mathtt{O}, \mathtt{S_j})$ both contain (i) the same relation $\mathtt{R_i}(\mathtt{O}, \dots)$ or (ii) the same attribute $\mathtt{A_i}(\mathtt{O}, \mathtt{D})$. Note that they cannot contain the same part since two classes such that one is a descendant of the other in the class hierarchy never have a part with the same name. In each of the above cases, the shared relation (or attribute) can be pushed into each subclass (O, S_j) by distributing $\rho(\mathtt{O}, \mathtt{R_i}, \mathbf{W})$ (or $\alpha(\mathtt{A_i}, \mathtt{C_i}, \mathbf{W})$) over the subclass sum and into each subclass (this can be repeated for multiple levels of the class hierarchy). This makes the product over relations (attributes) in $\phi(\mathtt{O}, \mathtt{S_j}, \mathbf{W})$ non-decomposable, but does not affect the decomposability of any other objects. Now, $\phi(\mathtt{O}, \mathtt{S_j}, \mathbf{W})$ can be decomposed, as follows. For (i), the product over relations in $\phi(\mathtt{O}, \mathtt{S_j}, \mathbf{W})$ now contains the non-decomposable factor $F(\mathtt{R_i}) = \rho(\mathtt{O}, \mathtt{R_i}, \mathbf{W}) \cdot \rho'(\mathtt{O}, \mathtt{R_i}, \mathbf{W})$, where ρ' was pushed down from $(\mathtt{O}, \mathtt{C})$. However, $F(\mathtt{R_i}) = (e^{w_i}[\mathtt{R_i}] + [\neg \mathtt{R_i}]) \cdot (e^{w'_i}[\mathtt{R_i}] + [\neg \mathtt{R_i}]) = e^{w_i + w'_i}[\mathtt{R_i}] + [\neg \mathtt{R_i}]$ since $[a]^2 = [a]$ and $[a][\neg a] = 0$ for a literal a. Thus, $F(\mathtt{R_i})$ is simply $\rho(\mathtt{O}, \mathtt{R_i}, \mathbf{W})$ with weight $w_i + w'_i$ for $\mathtt{R_i}$, which results in the same decomposable SPN structure with a different weight. For (ii), let the weight function for attribute $\mathtt{A_i}$ of class $\mathtt{C}$ with domain $\mathbf{D}_i$ be $\mathbf{u}_i$ and the weight function from the attribute that was pushed down be $\mathbf{u}'_i$. To render this decomposable, simply replace $\mathbf{u}_i$ with $\mathbf{u}_i \odot \mathbf{u}'_i$, the element-wise product of the two weight functions. Again, decomposability is

achieved simply by updating the weight functions. Decomposing (i) and (ii) each adds only a linear number of edges to the original non-decomposable SPN, so the size of the corresponding decomposable SPN is $|\mathcal{K}|$. Thus, from the sum-product theorem, computing the partition function of TPKB $\mathcal{K}$ takes time linear in $|\mathcal{K}|$. □

3.6 Empirical Evaluation of LearnSPF

I evaluated LearnSPF on the task of learning a nonconvex decomposable min-sum function (MSF) from a training set of solutions of instances of a highly-multimodal test function comprised of a sum of terms. Learning an MSF, instead of just a sum of terms, learns the general mathematical form of the optimization problem in such a way that the resulting learned function is tractable, whereas the original sum of terms is not. The test function I used is a variant of the Rastrigin function [144], a standard highly-multimodal test function for global optimization consisting of a sum of multi-dimensional sinusoids in quadratic basins. The function, $F_{\mathbf{X}}(\mathbf{Y}) = F(\mathbf{Y}; \mathbf{X})$, has parameters $\mathbf{X}$, which determine the dependencies between the variables $\mathbf{Y}$ and the location of the minima. To test LearnSPF, I sampled a dataset of function instances $T = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{m}$ from a distribution over $\mathcal{X} \times \mathcal{Y}$, where $\mathbf{y}^{(i)} = \arg\min_{\mathbf{y} \in \mathcal{Y}} F_{\mathbf{x}^{(i)}}(\mathbf{y})$.

LearnSPF partitioned variables $\mathbf{Y}$ based on the connected components of a graph containing a node for each $Y_i \in \mathbf{Y}$ and an edge between two nodes only if Y_i and Y_j were correlated, as measured by Spearman rank correlation. Instances were clustered by running k-means on the values $\mathbf{y}^{(i)}$. For this preliminary test, LearnSPF did not learn the leaf functions of the learned min-sum function (MSF) $M(\mathbf{Y})$; instead, when evaluating or minimizing a leaf node in M, the test function was evaluated or minimized with all variables not in the scope of the leaf node fixed to 0 (none of the optima were positioned at 0). This corresponds to having perfectly learned leaf nodes if the scopes of the leaf nodes accurately reflect the decomposability of F, otherwise a large error is incurred. I did this to study the effect of learning the decomposability structure in isolation from the error that would be incurred by also learning leaf nodes. The function used for comparison is also

learned perfectly. Thresholds t and v were set to 30 and 2, respectively.

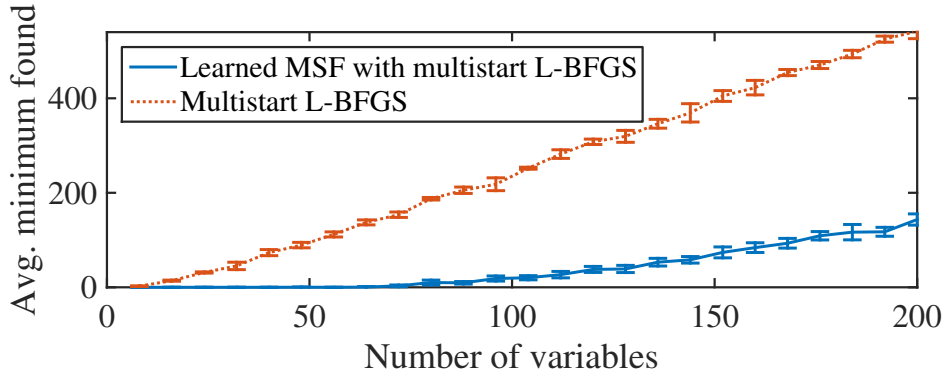


Figure 3.1: Plot of the average minimum found over 20 samples of the test function as a function of the number of variables of the test function when using multistart L-BFGS optimization directly and when using multistart L-BFGS on the learned MSF, with standard error bars. Each function was optimized for the same amount of time and has global minimum 0.

The dataset was split into 300 training samples and 20 test samples, where $\min_{\mathbf{y}} F_{\mathbf{x}^{(i)}}(\mathbf{Y}) = 0$ for all i for comparison purposes. After training, $\mathbf{y}_M = \arg\min_{\mathbf{y}} M(\mathbf{Y})$ was computed for each function in the test set by first minimizing each leaf function (with respect to only those variables in the scope of the leaf function) with multi-start L-BFGS [97] and then performing an upward and a downward pass in M. Figure 3.1 shows the result of minimizing the learned MSF M and then evaluating the test function at $\mathbf{y}_M$ (blue line) compared to running multi-start L-BFGS directly on the test function and reporting the minimum found (red line). Both optimizations are run for the same (fixed) amount of time (one minute per test sample). Figure 3.1 shows that LearnSPF accurately learned the decomposition structure of the test function, allowing it to find much better minima when optimized, since optimizing many small functions at the leaves requires exploring exponentially fewer modes than optimizing the full function.

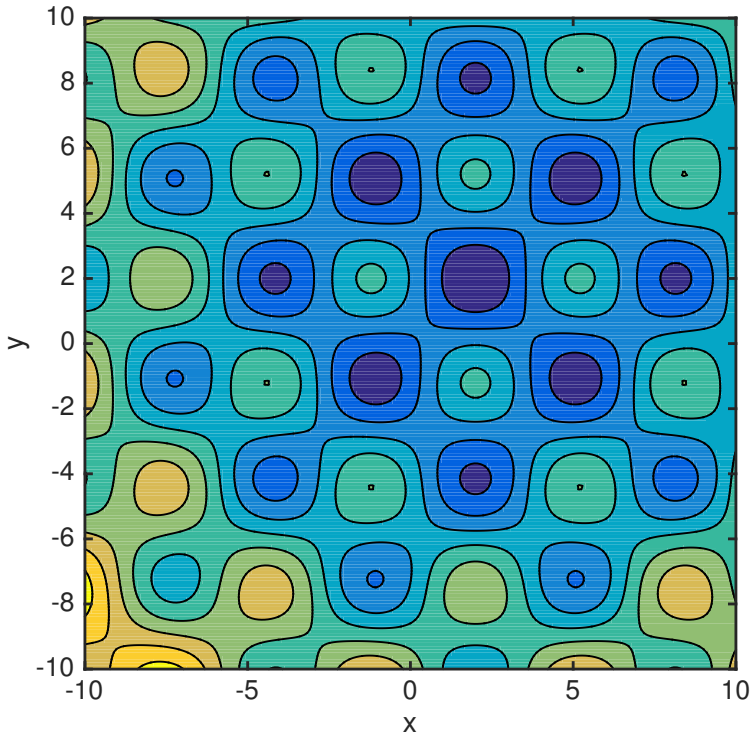


Figure 3.2: Contour plot of the 2-D nonconvex Rastrigin function. Dark blue indicates smaller values and yellow indicates larger values.

3.6.1 Experiment Details

All experiments were run on the same MacBook Pro with 2.2 GHz Intel Core i7 processor with 16 GB of RAM. Each optimization was limited to a single thread.

The non-separable variant of the Rastrigin function [144] used on pairs of variables is defined as

$$f^R_{x_i,x_j}(Y_i, Y_j) = c_0[(Y_i - x_i)^2 - (Y_j - x_j)^2] + \\ c_1 - c_1 \cos(Y_i - x_i) \cos(Y_j - x_j),$$

which has a global minimum at $\mathbf{y}^* = (x_i, x_j)$ with value $f^R_{\mathbf{x}}(\mathbf{y}^*) = 0$. The constants c_0 and c_1 control the shape of the quadratic basin and the amplitude of the sinusoids, respectively.

For my experiments, I used $c_0 = 0.1$ and $c_1 = 20$. Figure 3.2 shows a contour plot of f^R.

Omitting the parameters $\mathbf{x}$ from $f^R_{\mathbf{x}}$ for simplicity, the full test function for $n = 4m$ variables is defined as

$$F_{\mathbf{x}}(\mathbf{Y}) = \sum_{i=0}^{m} f^R(Y_{4i}, Y_{4i+k}) + f^R(Y_{4i+3}, Y_{4i+3-k}),$$

where $k = 1$ with probability 0.5 and $k = 2$ otherwise. This creates a function that is non-decomposable between each pair of variables (Y_{4i}, Y_{4i+k}) and (Y_{4i+3}, Y_{4i+3-k}). For the simplest case with $n = 4$ variables, if $k = 1$ then pairs (Y_0, Y_1) and (Y_2, Y_3) are non-decomposable. Alternatively, if $k = 2$ then pairs (Y_0, Y_2) and (Y_1, Y_3) are non-decomposable. The global minimum (x_i, x_j) for each function $f^R_{x_i,x_j}(Y_i, Y_j)$ was sampled uniformly over an interval of length 2 from the line $Y_i = Y_j$ with zero-mean additive Gaussian noise $(\sigma = 0.1)$. Thus, each instance of $F_{\mathbf{x}}$ is highly nonconvex and is decomposable with respect to certain variables and not with respect to others. For a set of instances of $F_{\mathbf{x}}$, there is structure in the decomposability between variables, but different instances have different decomposability structure, so LearnSPF must first group those function instances that have similar decomposability structure and then identify that structure in order to learn a min-sum function that is applicable to any instance in the training data.

3.7 Conclusion

This chapter developed a novel foundation for learning tractable representations in any semiring based on the sum-product theorem, a simple tractability condition for all inference problems that are summation on a semiring. With it, I developed a general inference algorithm and an algorithm for learning tractable representations in any semiring. I demonstrated the power and generality of my approach by applying it to learning a nonconvex function that can be optimized in polynomial time, a new type of structured prediction problem. I showed empirically that the learned min-sum function greatly outperforms a continuous function learned without regard to the cost of optimizing it. I

also showed that the sum-product theorem specifies an exponentially weaker condition for tractability than low treewidth and that its corollaries include many previous results in the literature, as well as a number of novel results. Many of the results in this chapter also appeared in Friesen and Domingos [45].

The work presented in this chapter only scratches the surface of applications of the sum-product theorem. First, while many semirings and inference problems have been discussed above, there remain many more semirings and domains in artificial intelligence and machine learning that would also benefit from the sum-product theorem (or already have, in the case of compositional kernel machines [49]) and from directly learning tractable models from data, such as learning sets of rules [117]. Second, investigating and extending SumSPF and Decompose to better handle inference in non-decomposable SPFs is important for extending SPFs and SPF learning to even more challenging classes of functions. I discuss one example of this in the next chapter, in which I present the RDIS algorithm for nonconvex optimization. However, there are many other interesting and important problems, such as integration, that would similarly benefit from extensions of these algorithms. Third, there has been much work in recent years on learning the structure of SPNs and similar models from data (e.g., Adel et al. [2], Gens and Domingos [48], Rooshenas and Lowd [120]). Extending these approaches to learning the structure of SPFs would benefit all of the domains and problems discussed above. Fourth, while the model classes that result from the sum-product theorem and decomposability are quite expressive, there remain problems and models for which they are insufficient, such as for scene understanding with arbitrary region shapes. However, by combining decomposability with other types of structure, such as symmetry group theory [102], it is possible to define even more expressive models. In Chapter 5, for example, I present submodular field grammars, which generalize SPNs by exploiting both decomposability and submodularity. Finally, the ideas underlying the sum-product theorem extend beyond SPFs and can be used to directly develop efficient algorithms for other problems, such as learning. In Chapter 6, I build on these ideas to develop a novel algorithm for learning deep neural networks that cannot be

learned via backpropagation.

Chapter 4

RECURSIVE DECOMPOSITION FOR NONCONVEX OPTIMIZATION

4.1 Introduction

Artificial intelligence (AI) systems that interact with the real world often have to solve continuous optimization problems. For convex problems, which have no local optima, many sophisticated algorithms exist. However, most continuous optimization problems in AI and related fields are nonconvex, and often have an exponential number of local optima. For these problems, the standard solution is to apply convex optimizers with multi-start and other randomization techniques [129], but in problems with an exponential number of optima these typically fail to find the global optimum in a reasonable amount of time. Branch and bound methods can also be used, but scale poorly due to the curse of dimensionality [106].

In this chapter, I propose that nonconvex optimization problems can instead be approached using problem decomposition techniques, which have a long and successful history in AI for solving discrete problems (e.g, [8, 12, 34, 37, 126, 127]). By repeatedly decomposing a problem into independently solvable subproblems, these algorithms can often solve in polynomial time problems that would otherwise take exponential time. The difficulty in nonconvex optimization is the combinatorial structure of the modes, which convex optimization and randomization are ill-equipped to deal with, but problem decomposition techniques are well suited to. I thus propose a novel nonconvex optimization algorithm, which uses recursive decomposition to handle the hard combinatorial core of the problem, leaving a set of simpler subproblems that can be solved using standard continuous optimizers.

The main challenges in applying problem decomposition to continuous problems are (a) extending existing approaches to handle continuous values, and (b) identifying a flexible and general type of structure that enables decomposing the underlying function. I do the former by embedding a continuous optimizer within the problem decomposition search, in a manner reminiscent of satisfiability modulo theory solvers [38], but for optimization not decision problems. I do the latter by observing that many continuous objective functions are approximately locally decomposable, in the sense that setting a subset of the variables causes the rest to break up into subsets that can be optimized nearly independently. This is particularly true when the objective function is a sum of terms over subsets of the variables, as is typically the case. A number of continuous optimization techniques employ a static, global decomposition (e.g., block coordinate descent [109] and partially separable methods [53]), but many problems only decompose locally and dynamically, which the algorithm presented in this chapter accomplishes.

For example, consider protein folding [6, 10], the process by which a protein, consisting of a chain of amino acids, assumes its functional shape. The computational problem is to predict this final conformation by minimizing a highly nonconvex energy function consisting mainly of a sum of pairwise distance-based terms representing chemical bonds, electrostatic forces, etc. Physically, in any conformation, an atom can only be near a small number of other atoms and must be far from the rest; thus, many terms are negligible in any specific conformation, but each term is non-negligible in some conformation. This suggests that sections of the protein could be optimized independently if the terms connecting them were negligible but that, at a global level, this is never true. However, if the positions of a few key atoms are set appropriately then certain amino acids will never interact, making it possible to decompose the problem into multiple independent subproblems and solve each separately. A local recursive decomposition algorithm for continuous problems can do exactly this.

In this chapter, I first define local decomposability and then present my algorithm, RDIS, which (asymptotically) finds the global optimum of a nonconvex function by Recursively

Decomposing the function into locally Independent Subspaces. RDIS is an instance of the SumSPF algorithm (Section 3.3) for nonconvex optimization (Section 3.5.4), in which values are chosen via a problem-specific subspace optimizer and variables are chosen by graph partitioning. In my analysis, I show that RDIS achieves an exponential speedup versus traditional techniques for nonconvex optimization such as gradient descent with restarts and grid search (although the complexity remains exponential, in general). This result is supported empirically, as RDIS significantly outperforms standard nonconvex optimization algorithms on three challenging domains: structure from motion, highly-multimodal test functions, and protein folding.

4.2 Recursive Decomposition for Continuous Optimization

In unconstrained continuous optimization, the goal is to minimize an objective function $f(X)$ over the variables $X \in \mathbb{R}^n$ for functions $f : \mathbb{R}^n \to \mathbb{R}$ that are continuously differentiable and have a nonempty optimal set $\mathbf{x}^*$ with optimal value $f^* = f(\mathbf{x}^*) > -\infty$. Let $\mathcal{I} = \{1, \ldots, n\}$ denote the indices of X, let $C \subseteq \mathcal{I}$ denote a subset of the indices, let $X_C \in \mathbb{R}^{|C|}$ denote the restriction of variables X to the indices in C, and let $\mathbf{c} \in \text{domain}(X_C)$ denote a partial assignment of X where only the variables corresponding to the indices in C are assigned values. I define $X|_{\mathbf{c}} \in \mathbb{R}^{n-|C|}$ to be the subspace where those variables with indices in C are set to the values in $\mathbf{c}$ (i.e., $X|_{\mathbf{c},i} = \mathbf{c}_i$ for some $\mathbf{c}$ and for all $i \in C$). Given $U = \mathcal{I} \backslash C$ and partial assignment $\mathbf{c}$, then, with a slight abuse of notation, I define the restriction of the function to the domain $X|_{\mathbf{c}}$ as $f|_{\mathbf{c}}(X_U)$. In the following, I directly partition X instead of discussing the partition of $\mathcal{I}$ that induces it, for simplicity. Since the focus of this chapter is on minimizing a nonconvex function, I will define everything with respect to the min-sum semiring; however, it is possible to generalize much of the work presented in this chapter to other semirings using the ideas presented in Chapter 3.

4.2.1 Local Decomposability

A function is fully decomposable (aka. separable) if it can be expressed as $f(X) = \sum_{i=1}^{n} g_i(X_i)$. Such functions are easy to optimize, since they decompose with respect to minimization; i.e., $\min_X f(X) = \sum_{i=1}^{n} \min_{X_i} g_i(X_i)$. Conversely, decomposable nonconvex functions that are optimized without first decomposing them require exponentially more exploration to find the global optimum than the decomposed version. For example, let M_f be the set of modes of f and let M_i be the modes of each g_i. Knowing that f is decomposable allows us to optimize each g_i independently, giving $|M_f| = \sum_{i=1}^{n} |M_i|$ modes to explore. However, if instead f was optimized directly, the optimizer would have to explore $\prod_{i=1}^{n} |M_i|$ modes, which is exponential in n. Unfortunately, fully decomposable functions like f are rare, as variables generally appear in multiple terms with many different variables and thus the minimization does not trivially distribute. However, decomposition can still be achieved if the function exhibits global or local decomposability structure.

Definition 4.1.

(a) *$f(X)$ is **globally decomposable** if there exists a partition $\{X_C, X_{U_1}, X_{U_2}\}$ of X such that $f|_{\mathbf{c}}(X_{U_1}, X_{U_2}) = f_1|_{\mathbf{c}}(X_{U_1}) + f_2|_{\mathbf{c}}(X_{U_2})$ for every partial assignment $\mathbf{c}$ of X_C.*

(b) *$f(X)$ is **locally decomposable** in the subspace $X|_{\mathbf{c}}$ if there exists a partition $\{X_C, X_{U_1}, X_{U_2}\}$ of X and a partial assignment $\mathbf{c}$ such that $f|_{\mathbf{c}}(X_{U_1}, X_{U_2}) = f_1|_{\mathbf{c}}(X_{U_1}) + f_2|_{\mathbf{c}}(X_{U_2})$.*

(c) *$f(X)$ is **approximately locally decomposable** in a neighbourhood of the subspace $X|_{\mathbf{c}}$ if there exists a partition $\{X_C, X_{U_1}, X_{U_2}\}$ of X, partial assignments $\mathbf{c}, \mathbf{d}$ of X_C, and $\delta, \epsilon \geq 0$ such that if $||\mathbf{d} - \mathbf{c}|| \leq \delta$ then $|f|_{\mathbf{d}}(X_{U_1}, X_{U_2}) - [f_1|_{\mathbf{d}}(X_{U_1}) + f_2|_{\mathbf{d}}(X_{U_2})]| \leq \epsilon$.*

Global decomposability (Definition 4.1a), while the easiest to exploit, is also the least prevalent. Local decomposability, which may initially appear limited, subsumes global decomposability while also allowing different decompositions throughout the space, making it strictly more general. Similarly, approximate local decomposability subsumes local decomposability. In protein folding, for example, two amino acids may be pushed either close together or far apart for different configurations of other amino

acids. In the latter case, they can be optimized independently because the terms connecting them are negligible. Thus, for different partial configurations of the protein, different approximate decompositions are possible. The independent subspaces that result from local decomposition can themselves exhibit local decomposability structure, allowing them to be decomposed in turn. If an algorithm exploits local decomposability effectively, it never has to perform the full combinatorial optimization. Local decomposability does not need to exist everywhere in the space, just in the regions being explored. For convenience, I only refer to local decomposability below, unless the distinction between global or (approximate) local decomposability is relevant.

One method for achieving local decomposability is via (local) simplification. I say that $f_i(X_C, X_U)$ is *(approximately locally) simplifiable* in the subspace $X|_{\mathbf{c}}$ defined by partial assignment $\mathbf{c}$ if $\overline{f_i}|_{\mathbf{c}}(X_U) - \underline{f_i}|_{\mathbf{c}}(X_U) \leq 2\epsilon$ for a given $\epsilon \geq 0$, where $\overline{h}(X)$ and $\underline{h}(X)$ denote the upper and lower bounds of $h(X)$, respectively. Similarly, $f(X)$ is *(approximately locally) simplified* in the subspace $X|_{\mathbf{c}}$ defined by partial assignment $\mathbf{c}$ if all simplifiable terms $f_i|_{\mathbf{c}}(X_U)$ are replaced by the constant $k_i = \frac{1}{2}\left[\ \overline{f_i}|_{\mathbf{c}}(X_U) + \underline{f_i}|_{\mathbf{c}}(X_U)\right]$ for a given $\epsilon \geq 0$. For a function that is a sum of terms, local decomposition occurs when some of these terms simplify in such a way that the minimization can distribute over independent groups of terms and variables (like component decomposition in Relsat [12] or in the protein folding example above). Given that there are m terms in the function, the maximum possible error in the simplified function versus the true function is $m \cdot \epsilon$. However, this would require all terms to be simplified and their true values to be at one of their bounds, which is extremely unlikely; rather, errors in different terms often cancel, and the simplified function tends to remain accurate. Note that ϵ induces a tradeoff between acceptable error in the function evaluation and the computational cost of optimization, since a simplified function has fewer terms and thus evaluating it and computing its gradient are both cheaper. While the above definition is for sums of terms, the same mechanism applies to functions that are products of (non-negative) factors – i.e., functions in the min-product semiring – although error grows multiplicatively here.

4.2.2 *The RDIS Algorithm*

RDIS is an optimization method that explicitly finds and exploits local decomposition. Pseudocode is shown in Algorithm 4, with subroutines explained in the text. At each level of recursion, RDIS chooses a subset of the variables $X_C \subseteq X$ (inducing a partition $\{X_C, X_U\}$ of X) and assigns them values $\mathbf{c}$ such that the simplified objective function $f|_{\mathbf{c}}(X_U)$ decomposes into multiple (approximately) independent sub-functions $\{f_i|_{\mathbf{c}}(X_{U_i})\}_{i=1}^{k}$, where $\{X_{U_1}, \ldots, X_{U_k}\}$ is a partition of X_U and $1 \leq k \leq n$. RDIS then recurses on each sub-function, globally optimizing it conditioned on the assignment $X_C = \mathbf{c}$. When the recursion completes, RDIS uses the returned optimal values of X_U (computed conditioned on $\mathbf{c}$) to choose a new value $\mathbf{c}$ for X_C and then simplifies, decomposes, and optimizes the function again. This repeats until a heuristic stopping criterion is satisfied.

Algorithm 4 Recursive Decomposition into locally Independent Subspaces (RDIS).

Input: Function $f(X)$ over variables X, initial state $\mathbf{x}^0$, subspace optimizer S, and approximation error ϵ.
Output: (Approximate) global minimum f^* at state $\mathbf{x}^*$.

function RDIS($f(X), X, \mathbf{x}^0, S, \epsilon$)
  $X_C \leftarrow$ ChooseVars(X) *// variable selection (choose subset $X_C \subseteq X$)*
  set $X_U \leftarrow X \backslash X_C$, $f^* \leftarrow \infty$, and $\mathbf{x}^* \leftarrow \mathbf{x}^0$
  repeat
    partition $\mathbf{x}^*$ into partial assignments $\{\mathbf{c}^*, \mathbf{u}^*\}$ of X_C and X_U
    $\mathbf{c} \leftarrow S(\ f|_{\mathbf{u}^*}(X_C),\ \mathbf{c}^*\ )$ *// value selection (choose value $\mathbf{c}$ of X_C)*
    $\hat{f}|_{\mathbf{c}}(X_U) \leftarrow$ simplify($f|_{\mathbf{c}}(X_U), \epsilon$) *// simplify f within subspace $X|_{\mathbf{c}}$*
    $\{\hat{f}_i(X_{U_i})\}_{i=1}^{k} \leftarrow$ decompose($\hat{f}|_{\mathbf{c}}(X_U)$) *// decompose $\hat{f}$ within subspace $X|_{\mathbf{c}}$*
    partition $\mathbf{u}^*$ into partial assignments $\{\mathbf{u}_i^*\}_{i=1}^{k}$ of $\{X_{U_i}\}$
    for i = 1, . . . , k **do**
      $\langle \hat{f}_i^*, \mathbf{u}_i \rangle \leftarrow$ RDIS($\hat{f}_i(X_{U_i}), X_{U_i}, \mathbf{u}_i^*, S, \epsilon$) *// recurse on each component*
    set $f_c^* \leftarrow \sum_{i=1}^{k} \hat{f}_i^*$ and $\mathbf{u} \leftarrow \cup_{i=1}^{k} \mathbf{u}_i$
    if $f_c^* < f^*$ **then**
      set $f^* \leftarrow f_c^*$ and $\mathbf{x}^* \leftarrow \mathbf{c} \cup \mathbf{u}$ *// record new minimum*
  until stopping criterion is satisfied
  return $\langle f^*, \mathbf{x}^* \rangle$

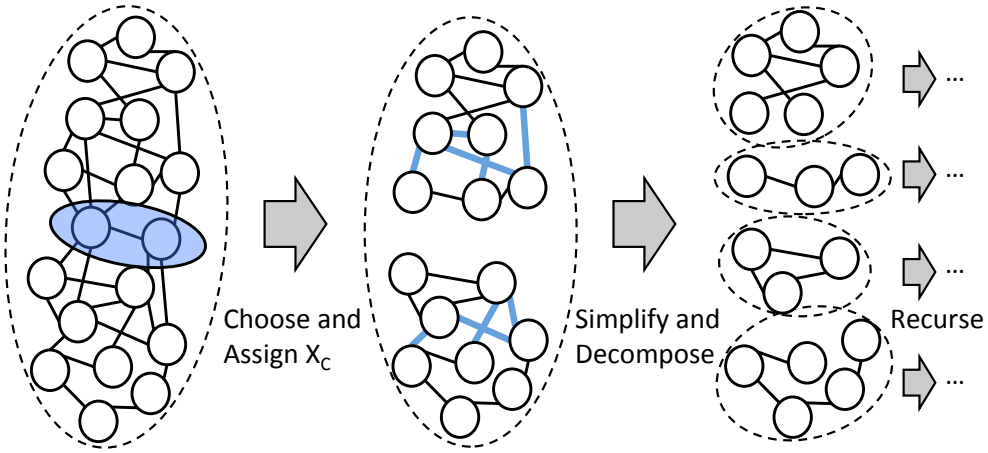


Figure 4.1: Visualization of RDIS decomposing the objective function. Vertices (circles) represent variables and edges connect each pair of variables in a term. Left: RDIS selects X_C (shaded oval). Middle: The function during simplification. Thick edges indicate simplifiable terms. Assigned variables are constant and have been removed. Right: The function after decomposition.

RDIS selects variables (line 2) heuristically, with the goal of choosing the smallest set of variables that enables the largest amount of decomposition, as this provides the largest computational gains. Specifically, RDIS uses a hypergraph partitioning algorithm to determine a small cutset that will decompose the graph; this cutset becomes the selected variables, X_C. Values $\mathbf{c}$ for variables X_C are determined (line 6) by calling a nonconvex subspace optimizer with the remaining variables (X_U) fixed to their current values $\mathbf{u}^*$. The subspace optimizer S is specified by the user and is customizable to the problem being solved. In my experiments I used multi-start versions of both conjugate gradient descent and Levenberg-Marquardt [109]. Restarts occur within line 6: if S converges without making progress then it restarts to a new value of X_C and runs until it reaches a local minimum.

To simplify the objective function (line 7), RDIS determines which terms are simplifiable (i.e., have sufficiently small bounds) and then simplifies (approximates) these by replacing them with a constant. These terms are not passed to the recursive calls. After variables have been assigned and the function simplified, RDIS locally decomposes (line 8) the

simplified function into independent sub-functions (components) that have no overlapping terms or variables and thus can be optimized independently, which is done by recursively calling RDIS on each. See Figure 4.1 for a visualization of this process. The recursion halts when ChooseVars selects all of X (i.e., $X_C = X$ and $X_U = \varnothing$), which occurs when X is small enough that the subspace optimizer can optimize $f(X)$ directly. At this point, RDIS repeatedly calls the subspace optimizer until the stopping criterion is met, which (ideally) finds the global optimum of $f|_{\mathbf{u}^*}(X_C) = f(X)$ since S is a nonconvex optimizer. The stopping criterion is user-specified, and depends on the subspace optimizer. If a multi-start descent method is used, termination occurs after a specified number of restarts, corresponding to a certain probability that the global optimum has been found. If the subspace optimizer is grid search, then the loop terminates after all values of X_C have been assigned. More subroutine details are provided in Section 4.4.

4.3 Analysis of RDIS

I now present analytical results demonstrating the benefits of RDIS versus standard algorithms for nonconvex optimization. Formally, I show that RDIS explores the state space in exponentially less time than the same subspace optimizer for a class of functions that are locally decomposable, and that it will (asymptotically) converge to the global optimum.

4.3.1 Complexity

RDIS begins by choosing a block of variables X_C. Assuming that this choice is made heuristically using the PaToH library for hypergraph partitioning, which is a multi-level technique, then the complexity of choosing variables is linear in the size of the hypergraph (see Trifunović [141], p. 81). Within the loop, RDIS chooses values for X_C, simplifies and decomposes the function, and finally recurses. Let the complexity of choosing values using the subspace optimizer be $g(d)$, where $|X_C| = d$, and let one call to the subspace optimizer be cheap relative to the number of variables n (e.g., computing the gradient of f with respect

to X_C or taking a step on a grid). Simplification requires iterating through the set of terms and computing bounds, so is linear in the number of terms, m. The connected components are maintained by a dynamic graph algorithm [61] which has an amortized complexity of $O(\log^2(|V|))$ per operation, where $|V|$ is the number of vertices in the graph. Finally, let the number of iterations of the loop $\xi(d)$ be a function of the dimension d since more dimensions generally require more restarts. The form of $\xi(d)$ depends on the subspace optimizer, but can be roughly interpreted as the number of modes of the sub-function $f|_{\mathbf{u}^*}(X_C)$ to explore times a constant factor.

Proposition 4.1. *If, at each level, RDIS chooses $X_C \subseteq X$ of size $|X_C| = d$ such that, for each selected value* $\mathbf{c}$*, the simplified function $\hat{f}|_{\mathbf{c}}(X_U)$ locally decomposes into $k > 1$ independent sub-functions $\{\hat{f}_i(X_{U_i})\}$ with equal-sized domains X_{U_i}, then the time complexity of RDIS is $O(\frac{n}{d}\xi(d)^{\log_k(n/d)})$.*

Proof. Assuming that m is of the same order as n, the recurrence relation for RDIS is $T(n) = O(n) + \xi(d)\left[g(d) + O(m) + O(n) + O(d\log^2(n)) + k\,T\left(\frac{n-d}{k}\right)\right]$, which can be simplified to $T(n) = \xi(d)\left[k\,T\left(\frac{n}{k}\right) + O(n)\right] + O(n)$. Noting that the recursion halts at $T(d)$, the solution to the above recurrence relation is then $T(n) = c_1\ (k\,\xi(d))^{log_k(n/d)} + c_2\,n\sum_{r=0}^{log_k(n/d)-1}\xi(d)^r$. which is $O\left((k\,\xi(d))^{\log_k(n/d)}\right) = O\left(\frac{n}{d}\xi(d)^{\log_k(n/d)}\right)$. □

Note that since RDIS uses hypergraph partitioning to choose variables, it will always decompose the remaining variables X_U unless they are fully connected. This is also supported by experimental results since if there were no decomposition, RDIS would not perform any better than the baselines.

From Proposition 4.1, the time complexity of RDIS for different subspace optimizers can be computed. Let the subspace optimizer be grid search (GS) over a bounded domain of width w with spacing δ in each dimension. Then the complexity of grid search is simply $O((w/\delta)^n) = O(s^n)$.

Proposition 4.2. *If the subspace optimizer is grid search, then $\xi(d) = (w/\delta)^d = s^d$, and the complexity of RDIS$_{GS}$ is $O(\frac{n}{d}s^{d\log_k(n/d)})$.*

Rewriting the complexity of grid search as $O(s^n) = O(s^{d(n/d)})$ shows that it is exponentially worse than the complexity of RDIS$_{GS}$ when decomposition occurs.

Now consider a descent method with random restarts (DR) as the subspace optimizer. Let the volume of the basin of attraction of the global minimum (the global basin) be l^n and the volume of the space be L^n, where a basin of attraction is defined as follows.

Definition 4.2. *The* basin of attraction *of a stationary point* $\mathbf{c}$ *is the set of points* $B \subseteq \mathbb{R}^n$ *for which the sequence generated by DR, initialized at* $\mathbf{x}^0 \in B$*, converges to* $\mathbf{c}$*.*

Then the probability of randomly restarting in the global basin is $(l/L)^n = p^n$. Since the restart behavior of DR is a Bernoulli process, the expected number of restarts to reach the global basin is $r = p^{-n}$, from the shifted geometric distribution. If the number of iterations needed to reach the stationary point of the current basin is τ then the expected complexity of DR is $O(\tau p^{-n})$. If DR is used within RDIS, then I obtain the following result.

Proposition 4.3. *If the subspace optimizer is DR, then the expected value of* $\xi(d)$ *is* τp^{-d}*, and the expected complexity of RDIS*$_{DR}$ *is* $O(\frac{n}{d}(\tau p^{-d})^{\log_k (n/d)})$.

Rewriting the expected complexity of DR as $O(\tau(p^{-d})^{n/d})$ shows that RDIS$_{DR}$ is exponentially more efficient than DR.

4.3.2 Convergence

Regarding convergence, RDIS with $\epsilon = 0$ converges to the global minimum given certain conditions on the subspace optimizer. For grid search, RDIS$_{GS}$ returns the global minimum if the grid is finite and has sufficiently fine spacing. For gradient descent with restarts, RDIS$_{DR}$ will converge to stationary points of $f(X)$ as long as steps by the subspace optimizer satisfy two technical conditions. The first is an Armijo rule guaranteeing sufficient decrease in f and the second guarantees a sufficient decrease in the norm of the gradient. These conditions are necessary to show that RDIS$_{DR}$ behaves like an inexact Gauss-Seidel method [18], and thus that each limit point of the generated sequence is a stationary point of $f(X)$.

In more detail, at each level of recursion, RDIS with $\epsilon = 0$ partitions X into $\{X_C, X_U\}$, sets values $\mathbf{c}$ using the subspace optimizer for X_C, globally optimizes $f|_{\mathbf{c}}(X_U)$ by recursively calling RDIS, and repeats. When the non-restart steps of the subspace optimizer satisfy two practical conditions (below) of sufficient decrease in (1) the objective function (a standard Armijo condition) and (2) the gradient norm over two successive partial updates, (i.e., conditions (3.1) and (3.3) of Bonettini [18]), then this process is equivalent to the 2-block inexact Gauss-Seidel method (2B-IGS) described in Bonettini [18] (see also Grippo and Sciandrone [54] and Cassioli et al. [23]), and each limit point of the sequence generated by RDIS is a stationary point of $f(X)$, of which the global minimum is one, and reachable through restarts.

Formally, let superscript r indicate the recursion level, with $0 \leq r \leq d$, with $r = 0$ the top, and recall that $X_U^{(r)} = \left\{X_C^{(r+1)}, X_U^{(r+1)}\right\}$ if there is no decomposition. The following proofs focus on the no-decomposition case for clarity; however, the extension to the decomposable case is trivial since each sub-function of the decomposition is independent. I denote applying the subspace optimizer to $f(X)$ until the stopping criterion is reached as $S_*(f, X)$ and a single call to the subspace optimizer as $S_1(f, X)$ and note that $S_*(f, X)$, by definition, returns the global minimum $\mathbf{x}^*$ and that repeatedly calling $S_1(f, X)$ is equivalent to calling $S_*(f, X)$.

For convenience, I restate conditions (3.1) and (3.3) from Bonettini [18] (without constraints) on the sequence $\{\mathbf{x}^{(k)}\}$ generated by an iterative algorithm on blocks X_i for $i = 1, \ldots, m$, respectively as

$$f(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_i^{(k+1)}, \ldots, \mathbf{x}_m^{(k)}) \leq f(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_i^{(k)} + \lambda_i^{(k)} \boldsymbol{d}_i^{(k)}, \ldots, \mathbf{x}_m^{(k)}), \tag{4.1}$$

where $\lambda_i^{(k)}$ is computed using Armijo line search and $\boldsymbol{d}_i^{(k)}$ is a feasible descent direction, and

$$\begin{aligned} ||\nabla_i f(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_i^{(k+1)}, \ldots, \mathbf{x}_m^{(k)})|| &\leq \eta ||\nabla_i f(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_{i-1}^{(k+1)}, \ldots, \mathbf{x}_m^{(k)})||, \quad i = 1, \ldots, m \\ ||\nabla_i f(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_i^{(k+1)}, \ldots, \mathbf{x}_m^{(k+1)})|| &\leq \eta ||\nabla_{i-1} f(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_{i-1}^{(k+1)}, \ldots, \mathbf{x}_m^{(k)})||, \quad i = 2, \ldots, m \end{aligned}$$

$$||\nabla_1 f(\mathbf{x}_1^{(k+2)}, \ldots, \mathbf{x}_i^{(k+1)}, \ldots, \mathbf{x}_m^{(k+1)})|| \leq \eta^{1-m} ||\nabla_m f(\mathbf{x}_1^{(k+1)}, \ldots, \mathbf{x}_m^{(k+1)})||, \tag{4.2}$$

where $\eta \in [0, 1)$ is a forcing parameter. See Bonettini [18] for further details. Any method that generates a sequence such that these conditions hold is an inexact Gauss-Seidel method and is guaranteed to converge to a critical point of $f(X)$ when $m = 2$. Let RDIS$_{DR}$ refer to RDIS$(f(X), X, \mathbf{x}^0, S = \text{DR}, \epsilon = 0)$. Then the probability with which RDIS$_{DR}$ will converge to the global minimum is as follows.

Proposition 4.4. *If the non-restart steps of RDIS satisfy conditions* (4.1) *and* (4.2)*, $\epsilon = 0$, the number of variables is n, the volume of the global basin is $v = l^n$, and the volume of the entire space is $V = L^n$, then RDIS$_{DR}$ returns the global minimum after t restarts, with probability $1 - (1 - (v/V))^t$.*

Proof.

Step 1. Given a finite number of restarts, one of which starts in the global basin, then RDIS$_{DR}$, with no recursion, returns the global minimum and satisfies conditions (4.1) and (4.2). This can be seen as follows.

At $r = 0$, RDIS$_{DR}$ chooses $X_C^{(0)} = X$ and $X_U^{(0)} = \varnothing$ and repeatedly calls $S_1(f^{(0)}, X_C^{(0)})$. This is equivalent to calling $S_*(f^{(0)}, X_C^{(0)}) = S_*(f, X)$, which returns the global minimum $\mathbf{x}^*$. Thus, RDIS$_{DR}$ returns the global minimum. Returning the global minimum corresponds to a step in the exact Gauss-Seidel algorithm, which is a special case of the IGS algorithm and by definition satisfies conditions (4.1) and (4.2).

Step 2. Now, if the non-restart steps of $S_1(f, X)$ satisfy conditions (4.1) and (4.2), then RDIS$_{DR}$ returns the global minimum. I show this by induction on the levels of recursion.

Base case. From Step 1, RDIS$_{DR}(f^{(d)}, X^{(d)})$ returns the global minimum and satisfies (4.1) and (4.2), since RDIS$_{DR}$ does not recurse beyond this level.

Induction step. Assume that RDIS$_{DR}(f^{(r+1)}, X^{(r+1)})$ returns the global minimum. I now show that RDIS$_{DR}(f^{(r)}, X^{(r)})$ returns the global minimum. RDIS$_{DR}(f^{(r)}, X^{(r)})$ first partitions $X^{(r)}$ into the two blocks $X_C^{(r)}$ and $X_U^{(r)}$ and then iteratively takes the following two steps: $\mathbf{c}^{(r)} \leftarrow S_1(f|_{\mathbf{u}^*}^{(r)}(X_C^{(r)}))$ and $\mathbf{u}^{(r)} \leftarrow$ RDIS$_{DR}(f|_{\mathbf{c}}^{(r)}(X_U))$. The first simply calls the subspace

optimizer on $X_C^{(r)}$. The second is a recursive call equivalent to RDIS$_{DR}(f^{(r+1)}, X^{(r+1)})$, which, from the inductive assumption, returns the global minimum $\mathbf{u}^{(r)} = \mathbf{u}^{(r)*}$ of $f|_{\mathbf{c}}^{(r)}(X_U)$ and satisfies conditions (4.1) and (4.2). For $S_1(f|_{\mathbf{u}^*}^{(r)}(X_C^{(r)}))$, RDIS$_{DR}$ will never restart the subspace optimizer unless the sequence it is generating converges. Thus, for each restart, since there are only two blocks and both the non-restart steps of $S_1(f|_{\mathbf{u}^*}^{(r)}(X_C^{(r)}))$ and the RDIS$_{DR}(f|_{\mathbf{c}}^{(r)}(X_U))$ steps satisfy conditions (4.1) and (4.2) then RDIS$_{DR}$ is a 2B-IGS method and the generated sequence converges to the stationary point of the current basin. At each level, after converging, RDIS$_{DR}$ will restart, iterate until convergence, and repeat for a finite number of restarts, one of which will start in the global basin and thus converge to the global minimum, which is then returned.

Step 3. Finally, since the probability of RDIS$_{DR}$ starting in the global basin is (v/V), then the probability of it not starting in the global basin after t restarts is $(1-(v/V))^t$. From above, RDIS$_{DR}$ will return the global minimum if it starts in the global basin, thus RDIS$_{DR}$ will return the global minimum after t restarts with probability $1-(1-(v/V))^t$. This completes the proof. □

For $\epsilon > 0$, there does not yet exist a proof of convergence, even in the convex case, since preliminary analysis indicates that there are rare corner cases in which the alternating aspect of RDIS, combined with the simplification error, can potentially result in a non-converging sequence of values; however, I have not observed this behavior in practice. Furthermore, my experiments clearly show $\epsilon > 0$ to be extremely beneficial, especially for large, highly-connected problems.

4.3.3 Related Algorithms

Beyond its discrete counterparts, RDIS is related to many well-known continuous optimization algorithms. If all variables are chosen at the top level of recursion, then RDIS simply reduces to executing the subspace optimizer. If one level of recursion occurs, then RDIS behaves similarly to alternating minimization algorithms (which also have

global convergence results [54]). For multiple levels of recursion, RDIS has similarities to block coordinate (gradient) descent algorithms (see Tseng and Yun [143] and references therein). However, what sets RDIS apart is that decomposition in RDIS is determined locally, dynamically, and recursively. The analysis and experiments in this chapter show that exploiting this can lead to substantial performance improvements.

With respect to Chapter 3, RDIS minimizes functions consisting of a sum of terms, which are flat min-sum functions with continuous leaves, as discussed previously in Section 3.5.4. To optimize these functions, RDIS repeatedly chooses a subset of the variables and assigns values to them in order to (locally) decompose the remaining variables. RDIS is thus an instance of SumSPF (Section 3.3) and can be equivalently interpreted as implicitly constructing a deep, decomposable min-sum function and "summing" it (in the min-sum semiring) online. Since the variables are continuous, RDIS cannot explore every possible value of a variable, so must settle for exploring only a subset of its domain, as chosen by the subspace optimizer.

4.4 RDIS Subroutines

In this section, I present the specific choices made for the subroutines in RDIS. Other choices are possible, but I leave investigations of these for future work.

4.4.1 Variable Selection

Many possible methods exist for choosing variables. For example, heuristics from satisfiability may be applicable (e.g., VSIDS [104]). However, RDIS uses hypergraph partitioning in order to ensure decomposition whenever possible. Hypergraph partitioning splits a graph into k components of approximately equal size while minimizing the number of hyperedges cut. To maximize decomposition, RDIS should choose the smallest block of variables that, when assigned, decomposes the remaining variables. This corresponds exactly to the set of edges cut by hypergraph partitioning on a hypergraph that has a vertex for each term and a

hyperedge for each variable that connects the terms that variable is in (note that this is the inverse of Figure 4.1). Formally, RDIS constructs a hypergraph $H = (V, E)$ with a vertex for each term, $\{n_i \in V : f_i \in f\}$ and a hyperedge for each variable, $\{e_j \in E : X_j \in X\}$, where each hyperedge e_j connects to all vertices n_i for which the corresponding term f_i contains the variable X_j. Partitioning H, the resulting cutset will be the smallest set of variables that need to be removed in order to decompose the hypergraph. And since assigning a variable to a constant effectively removes it from the optimization (and the hypergraph), the cutset is exactly the set that RDIS chooses on line 2. RDIS maintains such a hypergraph across iterations and uses the PaToH hypergraph partitioning library [24] to quickly find good, approximate partitions. A similar idea was used in Darwiche and Hopkins [36] to construct d-trees for recursive conditioning; however, they only apply hypergraph partitioning once at the beginning, whereas RDIS performs it at each level of the recursion.

While variable selection could be placed inside the loop, it would repeatedly choose the same variables because hypergraph partitioning is based on the graph structure. However, RDIS still exploits local decomposition because the variables and terms at each level of recursion vary based on local decomposability. In addition, edge and vertex weights could be set based on current bounds or other local information.

4.4.2 Value Selection

RDIS can use any nonconvex optimization subroutine to choose values, allowing the user to pick an optimizer appropriate to their domain. In my experiments, I use multi-start versions of both conjugate gradient descent and Levenberg-Marquardt, but other possibilities include Monte Carlo search, quasi-Newton methods, and simulated annealing. I experimented with both grid search and branch and bound, but found them practical only for easy problems. In my experiments, I have found it helpful to stop the subspace optimizer early, because values are likely to change again in the next iteration, making quick, approximate improvement more effective than slow, exact improvement.

4.4.3 *Simplification and Decomposition*

Simplification is performed by checking whether each term (or factor) is simplifiable and, if it is, setting it to a constant and removing it from the function. RDIS knows the analytical form of the function and uses interval arithmetic [56] as a general method for computing and maintaining bounds on terms to determine simplifiability. RDIS maintains the connected components of a dynamic graph [61] over the variables and terms (equivalent in structure to a factor or co-occurrence graph). Components in RDIS correspond exactly to the connected components in this graph. Assigned variables and simplified terms are removed from this graph, potentially inducing local decomposition.

4.4.4 *Caching and Branch & Bound*

RDIS' similarity to model counting algorithms suggests the use of component caching and branch and bound (BnB). I experimented with these and found them effective when used with grid search; however, they were not beneficial when used with descent-based subspace optimizers, which dominate grid-search-based RDIS on non-trivial problems. For caching, this is because components are almost never seen again, due to not re-encountering variable values, even approximately. For BnB, interval arithmetic bounds tended to be overly loose and no bounding occurred. My experience suggests that this is because the descent-based optimizer effectively focuses exploration on the minima of the space, which are typically close in value to the current optimum. However, I believe that future work on caching and better bounds would be beneficial.

4.4.5 *Execution Time*

Variable selection typically occupies only a tiny fraction of the runtime of RDIS, with the vast majority of RDIS' execution time spent computing gradients for the subspace optimizer. A small, but non-negligible amount of time is spent maintaining the component graph, but this is much more efficient than if RDIS were to recompute the connected components each

time, and the exponential gains from decomposition are well worth the small upkeep cost.

4.5 Empirical Evaluation of RDIS

I evaluated RDIS on three difficult nonconvex optimization problems with hundreds to thousands of variables: structure from motion, a high-dimensional sinusoid, and protein folding. Structure from motion is an important problem in vision, while protein folding is a core problem in computational biology. RDIS was run with a fixed number of restarts at each level, thus not guaranteeing that the global minimum was found. For structure from motion, I compared RDIS to that domain's standard technique of Levenberg-Marquardt (LM) [109] using the levmar library [98], as well as to a block-coordinate descent version (BCD-LM). In protein folding, gradient-based methods are commonly used to determine the lowest energy configuration of a protein, so I compared RDIS to conjugate gradient descent (CGD) and a block-coordinate descent version (BCD-CGD). CGD and BCD-CGD were also used for the high-dimensional sinusoid. Blocks were formed by grouping contextually-relevant variables together (e.g., in protein folding, variables from the same amino acid were not split into separate blocks). I also compared to ablated versions of RDIS. RDIS-RND uses a random variable selection heuristic and RDIS-NRR does not use any internal random restarts (i.e., it functions as a convex optimizer) but does have top-level restarts. In each domain, the optimizer I compare to was also used as the subspace optimizer in RDIS. All experiments were run on the same cluster. Each computer in the cluster was identical, with two 2.33GHz quad core Intel Xeon E5345 processors and 16GB of RAM. Each algorithm was limited to a single thread.

4.5.1 Structure from Motion

Structure from motion is the problem of reconstructing the geometry of a 3-D scene from a set of 2-D images of that scene. It consists of first determining an initial estimate of the parameters and then performing non-linear optimization to minimize the squared error between a set of 2-D image points and a projection of the 3-D points onto camera

models [142]. The latter, known as bundle adjustment, is the task I focus on here. In bundle adjustment, the goal is to minimize the error between a dataset of points in a 2-D image and a projection of fitted 3-D points representing a scene's geometry onto fitted camera models. The variables are the parameters of the cameras and the positions of the points and the cameras. This problem is highly structured in a global sense: cameras only directly interact with a subset of points, creating a bipartite graph structure that RDIS is able to exploit, but (nontrivial) local decomposability does not exist because the bounds on each term are too wide and tend to include ∞. The dataset used is the 49-camera, 7776-point data file from the Ladybug dataset [3], where the number of points is scaled proportionally to the number of cameras used (i.e., if half of the cameras were used, half of the points were included). There are 9 variables per camera and 3 variables per point.

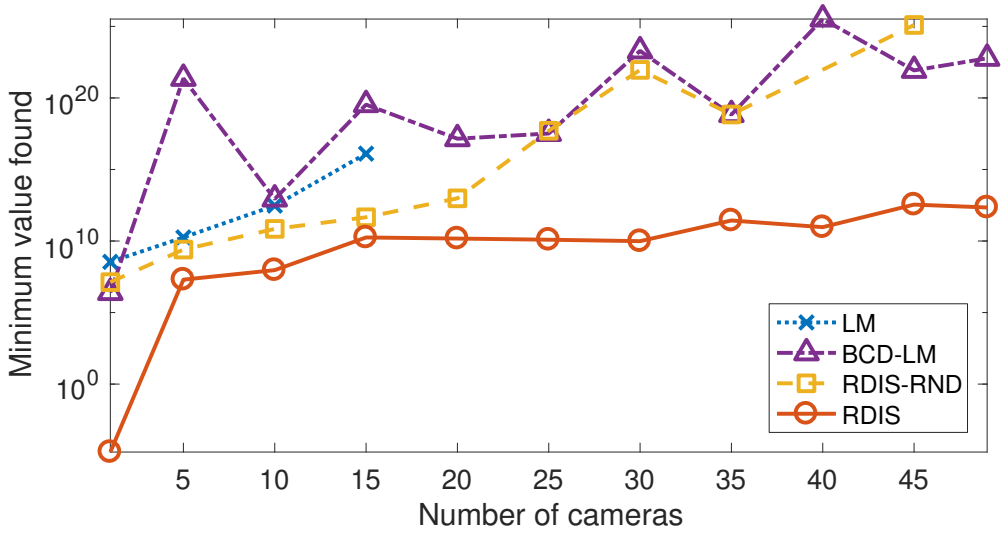


Figure 4.2: Minimum value found in five hours for increasing sizes of bundle adjustment problem (y-axis is log scale). Missing data points for LM are because it did not complete in the allotted time.

Figure 4.2 shows performance on bundle adjustment as a function of the size of the problem, with a log scale y-axis. Each point is the minimum error found after running each algorithm for five hours. Each algorithm is given the same set of restart states, but algorithms that converge faster may use more of these. Since no local decomposability is exploited,

Figure 4.2 effectively demonstrates the benefits of using recursive decomposition with intelligent variable selection for nonconvex optimization. Decomposing the optimization across independent subspaces allows the subspace optimizer to move faster, further, and more consistently, allowing RDIS to dominate the other algorithms. Missing points are due to algorithms not returning any results in the allotted time.

4.5.2 High-dimensional Sinusoid

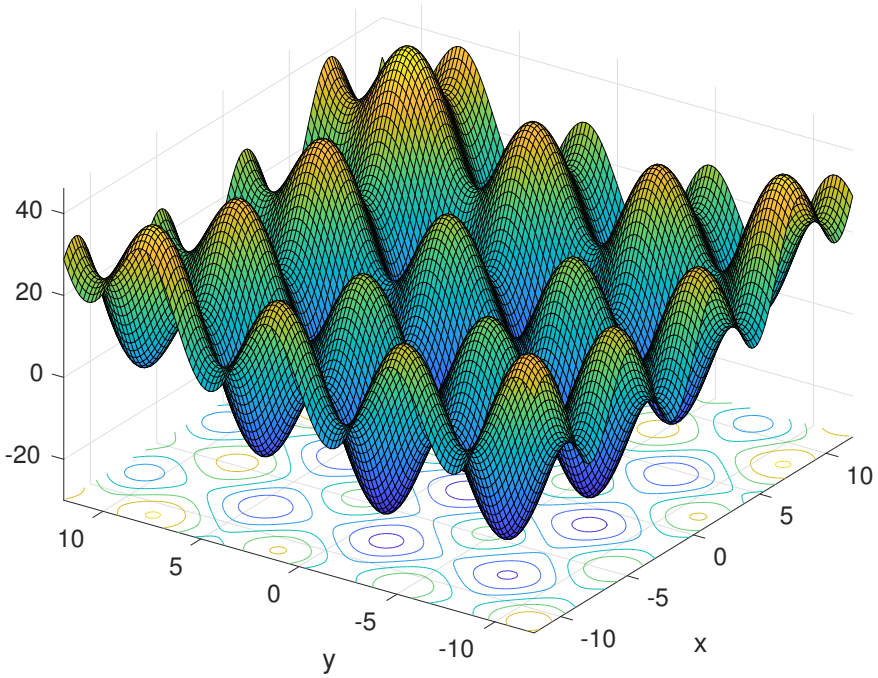


Figure 4.3: A 2-D example of the highly-multimodal test function optimized by RDIS.

The second domain is a highly-multimodal test function defined as a multidimensional sinusoid placed in the basin of a quadratic, with a small slope to make the global minimum unique. The arity of this function (i.e., the number of variables contained in each term) is controlled parametrically. Functions with larger arities contain more terms and dependencies, and thus are more challenging. The test function is defined as follows. Given a height h, a branching factor k, and a maximum arity a, define a complete k-ary tree of variables

of the specified height, with variable X_0 as the root. For all paths $p_j \in P$ in the tree of length $l_j \leq a$, with l_j even, define a term $t_{p_j} = \prod_{X_i \in p_j} \sin(X_i)$. The test function is then $f_{h,k,a}(X_0, \ldots, X_n) = \sum_{i=1}^{n} c_0 X_i + c_1 X_i^2 + c_2 \sum_P t_{p_j}$. The resulting function is a multidimensional sinusoid placed in the basin of a quadratic function parameterized by c_1, with a linear slope defined by c_0. The constant c_2 controls the amplitude of the sinusoids. For my tests, I used $c_0 = 0.6, c_1 = 0.1$, and $c_2 = 12$. A 2-D example of this function is shown in Figure 4.3. I used a tree height of $h = 11$, with branching factor $k = 2$, resulting in a function of 4095 variables. I evaluated each of the algorithms on functions with terms of arity $a \in \{4, 8, 12\}$, where a larger arity implies more complex dependencies between variables as well as more terms in the function. The functions for the three different arity levels had 16372, 24404, and 30036 terms, respectively. A small amount of local decomposability exists in this problem.

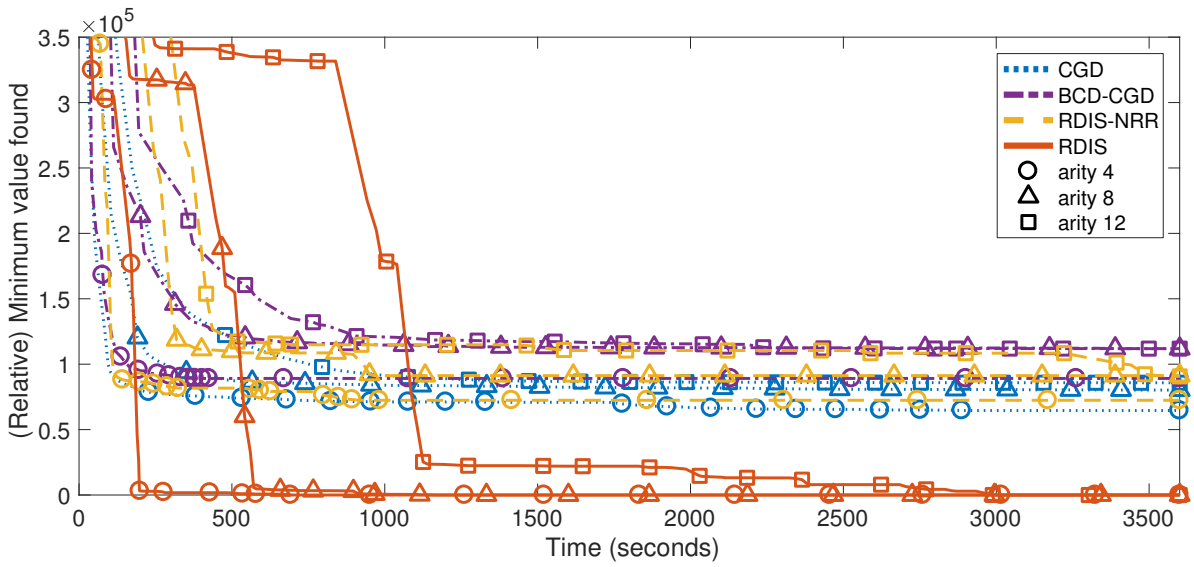


Figure 4.4: A comparison of the best minima found as a function of time for three different arities of the high-dimensional sinusoid.

Figure 4.4 shows the current best value found as a function of time. Each datapoint is from a single run of an algorithm using the same set of top-level restarts, although, again, algorithms that converge faster use more of these. RDIS outperforms all other algorithms, in-

cluding RDIS-NRR. This is due to the nested restart behavior afforded by recursive decomposition, which allows RDIS to effectively explore each subspace and escape local minima. The poor initial performance of RDIS for arities 8 and 12 is due to it being trapped in a local minimum for an early variable assignment while performing optimizations lower in the recursion. However, once the low-level recursions finish it escapes and finds the best minimum without ever performing a top level restart. The full optimization trajectories are shown in Figure 4.5.

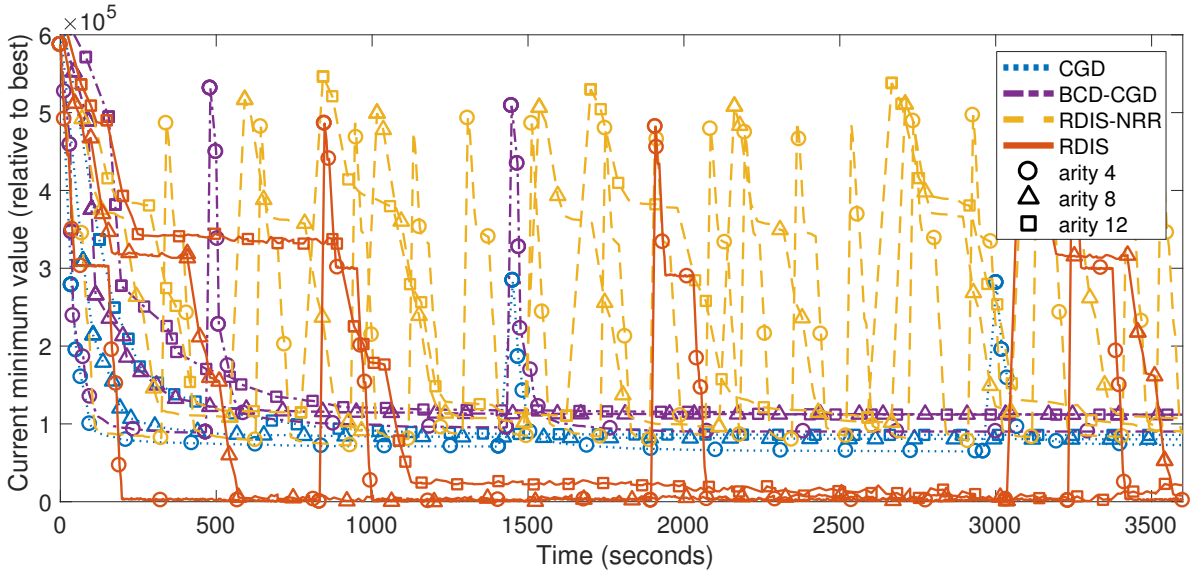


Figure 4.5: Optimization trajectories on the high-dimensional sinusoid. Sharp rises indicate restarts. Notably, RDIS-NRR restarts much more often than the other algorithms because decomposition allows it to move through the space much more efficiently. Without internal restarting it gets stuck at the same local minima as BCD-CGD and CGD. For arity 12, RDIS never performs a full restart and still finds the best minimum, despite using the same initial point as the other algorithms.

4.5.3 Protein Folding

The final domain is sidechain placement for protein folding with continuous angles between atoms. Protein folding [6, 10] is the process by which a protein, consisting of a long chain of amino acids, assumes its functional shape. The computational problem is to predict this final conformation given a known sequence of amino acids. This requires minimizing an

energy function consisting mainly of a sum of pairwise distance-based terms representing chemical bonds, hydrophobic interactions, electrostatic forces, etc., where, in the simplest case, the variables are the relative angles between the atoms. The optimal state is typically quite compact, with the amino acids and their atoms bonded tightly to one another and the volume of the protein minimized. Each amino acid is composed of a backbone segment and a sidechain, where the backbone segment of each amino acid connects to its neighbors in the chain, and the sidechains branch off the backbone segment and form bonds with distant neighbors. The sidechain placement task is to predict the conformation of the sidechains when the backbone atoms are fixed in place. Sidechain placement is equivalent to finding the MAP assignment of a continuous pairwise Markov random field [152].

Energies between amino acids are defined by the Lennard-Jones potential function, as specified in the Rosetta protein folding library [87]. The basic form of this function is $E_{LJ}(r) = \frac{A}{r^{12}} - \frac{B}{r^6}$, where r is the distance between two atoms and A and B are constants that vary for different types of atoms. The Lennard-Jones potential in Rosetta is modified slightly so that it behaves better when r is very large or very small. The full energy function is $E(\phi) = \sum_{\phi} E_{jk}(R_j(\chi_j), R_k(\chi_k))$, where R_j is an amino acid (also called a residue) in the protein, ϕ is the set of all torsion angles, and $\phi_i \in \chi_j$ are the angles for R_j. Each residue has between zero and four torsion angles that define the conformation of its sidechain, depending on the type of amino acid. The terms E_{jk} compute the energy between pairs of residues as $E_{jk} = \sum_{a_j} \sum_{a_k} E_{LJ}(r(a_j(\chi_j), a_k(\chi_k)))$, where a_j and a_k refer to the positions of the atoms in residues j and k, respectively, and $r(a_j, a_k)$ is the distance between the two atoms. The torsion angles define the positions of the atoms through a series of kinematic relations, which I do not detail here.

Significant local decomposability is present in this problem, which RDIS is able to exploit. Test proteins with sequence length between 300 and 600 were selected from the Protein Data Bank [15] such that the sequences of any two proteins did not overlap by more than 30%. The smallest (with respect to the number of terms) protein (ID 1) has 131 residues, 2282 terms, and 257 variables, while the largest (ID 21) has 440 residues, 9380

terms, and 943 variables. The average number of residues, terms, and variables is 334, 7110, and 682, respectively. The proteins with their IDs are as follows: (1) 4JPB, (2) 4IYR, (3) 4M66, (4) 3WI4, (5) 4LN9, (6) 4INO, (7) 4J6U, (8) 4OAF, (9) 3EEQ, (10) 4MYL, (11) 4IMH, (12) 4K7K, (13) 3ZPJ, (14) 4LLI, (15) 4N08, (16) 2RSV, (17) 4J7A, (18) 4C2E, (19) 4M64, (20) 4N4A, (21) 4KMA.

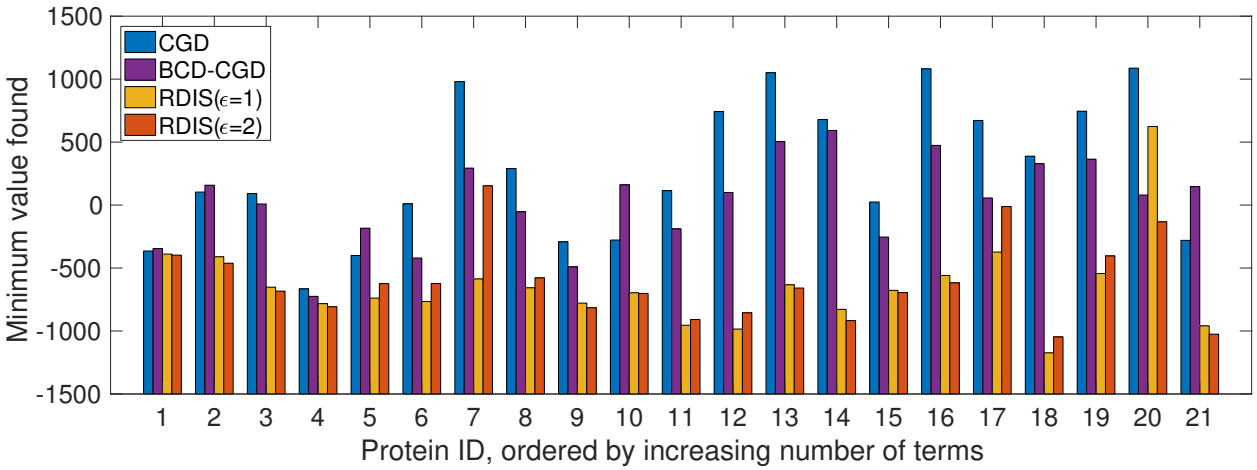


Figure 4.6: Minimum value (energy) found on 21 different proteins. Lower is better.

Figure 4.6 shows the results of combining all aspects of RDIS, including recursive decomposition, intelligent variable selection, internal restarts, and local decomposability on a difficult problem with significant local structure. Each algorithm is run for 48 hours on each of 21 proteins of varying sizes. RDIS is run with both $\epsilon = 1.0$ and $\epsilon = 2.0$ and both results are shown on the figure. RDIS outperforms CGD and BCD-CGD on all proteins, often by a large amount.

Figure 4.7 demonstrates the tradeoff between computation time and accuracy with respect to the approximation bound ϵ on RDIS for protein folding. It shows the performance of RDIS-NRR as a function of ϵ, where performance is measured both by minimum energy found and time taken. RDIS-NRR is used in order to remove the randomness associated with the internal restarts of RDIS, resulting in a more accurate comparison across multiple runs. Each point on the energy curve is the minimum energy found over the same 20

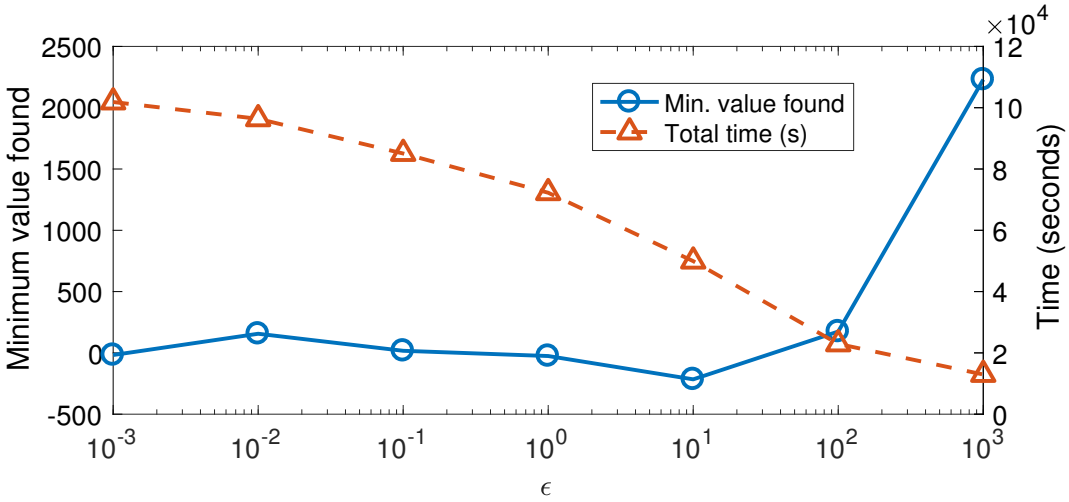


Figure 4.7: RDIS-NRR's minimum energy found and total optimization time taken to perform a fixed number (20) of restarts versus ϵ, on protein 3EEQ (ID 9). The x-axis is log scale.

restarts. Each point on the time curve is the total time taken for all 20 restarts. As ϵ increases, time decreases because more local decomposability structure is being exploited. In addition, minimum energy actually decreases initially. I attribute this to the smoothing caused by increased simplification, allowing RDIS to avoid minor local minima in the objective function.

4.6 Conclusion

This chapter proposed a new approach to solving hard nonconvex optimization problems based on recursive decomposition. RDIS decomposes the function into approximately locally independent sub-functions and then optimizes these separately by recursing on them. This results in an exponential reduction in the time required to find the global optimum. RDIS is an instance of SumSPF (Section 3.3) that selects variables using a hypergraph partitioning subroutine in order to achieve as much decomposition as possible and chooses values for these variables using a subspace optimizer in order to quickly minimize the function. In my experiments, I showed that RDIS consistently outperforms

competing methods for structure from motion, a high-dimensional sinusoid, and protein folding by explicitly and recursively decomposing the problem. The work presented in this chapter also appears in Friesen and Domingos [44].

Given the performance and flexibility of RDIS, there are many interesting directions for future work. For example, many other nonconvex optimization problems exhibit decomposability structure and can benefit from this approach, especially in other areas of science and engineering. While machine learning problems do not typically exhibit large amounts of decomposability as the model parameters tend to be fully connected, incorporating auxiliary variables or identifying latent factors can induce decomposability that RDIS-style methods can then exploit. In Chapter 6, I present an example of an RDIS-style method for deep learning that introduces target variables to decompose the learning problem. Other directions for future research include further analyzing RDIS' theoretical properties, developing new variable and value selection methods, extending RDIS to handle hard constraints, and using similar ideas for high-dimensional integration as also discussed in Section 3.5.4. Finally, the decomposability structure exploited by RDIS can be combined with other types of structure in order to develop even more expressive model classes. In the following chapter, I define submodular field grammars (SFGs), a new class of probabilistic model that combines submodular MRFs and sum-product networks (SPNs). By exploiting both decomposability and submodularity, approximate MAP inference in SFGs can be performed efficiently. For clarity, SFGs are presented in the context of image parsing, and are thus developed as a combination of stochastic image grammars – a type of SPN – and submodular MRFs. The connection to SPNs is made explicit in Section 5.3.1.

Chapter 5

SUBMODULAR FIELD GRAMMARS AND THEIR APPLICATION TO SCENE UNDERSTANDING

5.1 Introduction

Scene understanding is a challenging problem that requires simultaneously detecting, segmenting, and recognizing each object in the scene despite noise, distractors, and ambiguity. Fortunately, natural scenes possess inherent structure in the form of part-subpart relationships between objects. Such constituency relationships are well modeled by a grammar, which defines a set of production rules that specify the decomposition of objects into their parts. Natural language is the most common application of such grammars, but the compositional structure of natural scenes makes stochastic image grammars a natural candidate for representing distributions over images (see Zhu and Mumford [157] for a review). Importantly, natural language can be parsed efficiently with respect to a grammar because the number of possible split points for each production $A \rightarrow BC$ is linear in the length of the constituent corresponding to A. However, images cannot be parsed efficiently in this way because there are an exponential number of ways to split an image into arbitrarily-shaped subregions. As such, previous image grammar approaches could only ensure tractability by severely restricting the possible decompositions of each region either explicitly, for example by allowing only rectangular regions (e.g., Poon and Domingos [114]), or implicitly by sampling (e.g., Zhao and Zhu [154]).

Due to these restrictions, many approaches to scene understanding instead use Markov random fields (MRFs) over the pixels of an image (e.g., Shotton et al. [133], Gould et al. [50]) to capture some of the structure present in natural images while still permitting objects to have arbitrary shapes. Most MRFs for scene understanding are planar graphs that define

a probabilistic model over the labels of each pair of neighboring pixels, although some work has generalized these approaches to hierarchical MRFs [41, 82]. Unfortunately, while these models can improve labeling accuracy, as demonstrated by their continued use for scene understanding, their limited structure means that they can capture very little of the compositional structure present in natural images without requiring an exponential number of labels. Inference in MRFs is in general intractable [25], but is tractable under certain restrictions. For pairwise binary MRFs, if the energy is submodular [76], meaning that each pair of neighboring pixels prefers to have the same label – a natural assumption for images – then the exact MAP labeling of the MRF can be efficiently recovered with a graph-cut algorithm [20, 52]. For multi-label problems, a constant-factor approximation can be found efficiently using a move-making algorithm, such as α-expansion [21].

In this chapter, I combine the tractability and region-shape flexibility afforded by submodular MRFs with the high-level compositional structure of an image grammar. I associate with each production $A \to BC$ in the grammar a submodular MRF whose labels are the subconstituents (i.e., B, C) of that production. I call the resulting model a *submodular field grammar* (SFG). Finding the MAP labeling to split a region into arbitrarily-shaped subregions is now tractable and I exploit this to develop an efficient approximate algorithm for MAP parsing of images with SFGs. My algorithm, SFG-Parse, is an iterative move-making algorithm that provably converges to a local minimum of the energy and reduces to α-expansion in the case of a trivial grammar. Like other move-making algorithms, each step of SFG-Parse chooses the best move from an exponentially-large set of neighbors, thus overcoming many of the main issues with local minima [21].

Empirically, I compare SFG-Parse to belief propagation and α-expansion, where α-expansion provides a strong surrogate for the true global minimum, which is intractable to compute. I show that SFG-Parse parses images in exponentially less time than both of these while returning comparable minima. I also show promising improvements in accuracies when using an SFG in place of a standard MRF for scene understanding.

Like SFGs, associative hierarchical MRFs [93, 125] also define multi-level MRFs, but use

precomputed segmentations to set the regions of the non-terminal variables and thus do not permit arbitrary image regions. Neural parsing methods [131, 136] are grammar-like models for scene understanding, but use precomputed superpixels to segment the image and thus also do not permit arbitrary region shapes.

Although I present SFGs here in the context of scene understanding, they are a general and flexible model that is applicable anywhere grammars, SPNs, or MRFs are used, including social network modeling and probabilistic knowledge bases. The connection between SFGs and SPNs is made clear in Section 5.3.1.

5.2 Preliminaries

5.2.1 Submodular MRFs

A Markov random field (MRF) for scene understanding defines a probabilistic model

$$P(\mathbf{y}, \mathcal{I}) = \frac{1}{Z} \exp\left(-E(\mathbf{y}, \mathcal{I})\right) \tag{5.1}$$

over labeling $\mathbf{y} \in \mathcal{Y}^n$ and image $\mathcal{I}$, where $n = |\mathcal{I}|$ is the number of pixels, $\mathcal{Y}$ is the set of labels, which encode semantic classes such as Sky or Ground, and $Z = \sum_{\mathbf{y}' \in \mathcal{Y}^n} \exp(-E(\mathbf{y}', \mathcal{I}))$ is the partition function. MRFs for vision typically use pairwise energies $E(\mathbf{y}, \mathcal{I}) = \sum_{p \in \mathcal{I}} \theta_p(y_p) + \sum_{(p,q) \in \mathcal{I}} \theta_{pq}(y_p, y_q)$, where θ_p and θ_{pq} are the unary and pairwise energy terms for pixels p and edges (p, q), respectively; $\mathbf{y} = (y_0, \ldots, y_n)$; and, with a slight abuse of notation I say that $\mathcal{I}$ contains both the nodes and edges in the MRF over the image. For binary labels $\mathcal{Y} = \{Y_1, Y_2\}$, an MRF is submodular if its energy satisfies $\theta_{pq}(Y_1, Y_1) + \theta_{pq}(Y_2, Y_2) \leq \theta_{pq}(Y_1, Y_2) + \theta_{pq}(Y_2, Y_1)$ for all edges $(p, q) \in \mathcal{I}$. If the energy is submodular, the MAP labeling $\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} P(\mathbf{y}, \mathcal{I})$ can be computed exactly with a single graph cut in time $c(n)$, where $c(n)$ is worst-case low-order polynomial (the true complexity depends on the chosen min-cut/max-flow algorithm), but nearly linear time in practice [20, 21]. Thus, submodularity reduces the complexity of an optimization over 2^n states to nearly-linear time. While submodularity is useful for MAP inference, it also

captures the fact that neighboring pixels in natural images tend to have the same label (e.g., Sky pixels appear next to other Sky pixels), which means that in general the MAP labeling partitions the image into contiguous regions, where all pixels in a region have the same label.

5.2.2 Image Grammars

A context-free grammar (CFG) is a tuple $G = (N, \Sigma, R, S)$ containing a finite set of nonterminal symbols N; a finite set of terminal symbols Σ; a finite set of productions $R = \{v : X \to Y_1 \dots Y_k\}$ with head symbol $X \in N$ and subconstituent symbols $Y_i \in N \cup \Sigma$ for $i = 1 \dots k$; and a special start symbol $S \in N$ that does not appear on the right-hand side of any production. For scene understanding, a grammar for outdoor scenes might contain a production $S \to$ *Sky Ground*, which would partition the image into Sky and Ground subregions.

To extend CFGs to images, I introduce the notion of a region $\mathcal{R} \subseteq \mathcal{I}$, which specifies a subset of the pixels and can have arbitrary shape. A parse (tree) $t \in \mathcal{T}_G(\mathcal{I})$ for image $\mathcal{I}$ with respect to grammar G is a hierarchy of nodes $n = (v, \mathcal{R})$. Each node contains a production $v \in R$ and a corresponding image region $\mathcal{R} \subseteq \mathcal{I}$, where $\mathcal{T}_G(\mathcal{I})$ is the set of valid parse trees for $\mathcal{I}$ under G, which I will write as $\mathcal{T}$ to simplify notation. For each node $n = (v, \mathcal{R})$ in a parse tree, the regions of its children $\{c_i = (v_i, \mathcal{R}_i) : c_i \in \text{ch}(n)\}$ partition (segment) their parent's region such that $\mathcal{R} = \cup_i \mathcal{R}_i$ and $\cap_i \mathcal{R}_i = \varnothing$. Letting $v = X \to Y_1 \dots Y_k$, then this partition is equivalently defined by a labeling $\mathbf{y}^v \in \mathcal{Y}_v^{|\mathcal{R}|}$ where $\mathcal{Y}_v = \{Y_1, \dots, Y_k\}$, as there is a one-to-one correspondence between labelings and partitions of $\mathcal{R}$. Given a labeling for a production, the region of a subconstituent is simply the subset of pixels labeled as that subconstituent $\mathcal{R}_i = \{p : y_p^v = Y_i\}$ for any $i \in \{1, \dots, k\}$.

Finally, a stochastic image grammar defines a generative probabilistic model of images by associating with each nonterminal a categorical distribution over the productions of that nonterminal. The generative process is to sample a production of the current nonterminal from this distribution, starting with the start symbol S and the region containing the

entire image, and then to sample a partition of the current region into disjoint subregions – one for each subconstituent of the production. This process then recurses on each subconstituent-subregion pair, and terminates when a terminal symbol is produced, at which point the pixels for that region are generated. Formally, the probability of a parse $t \in \mathcal{T}$ of an image is

$$P(t, \mathcal{I}) = \prod_{(v,\mathcal{R}) \in t} P(v|\text{head}(v)) \cdot P(\mathbf{y}^v|v, \mathcal{R}), \tag{5.2}$$

where $P(\mathbf{y}^v|v, \mathcal{R})$ specifies the probability of each labeling $\mathbf{y}^v \in \mathcal{Y}_v^{|\mathcal{R}|}$ (partition) of $\mathcal{R}$. Note that the distribution over productions in (5.2) is the same categorical distribution as that used in PCFGs for natural language [68], but the distribution over segmentations is assumed to be uniform in PCFGs for natural language and is typically not made explicit. However, it is also this distribution that presents representational challenges, as a distribution $P(\mathbf{y}^v|v, \mathcal{R})$ is needed for each production and for each of the 2^n possible image regions. I address this in the following section.

5.3 Submodular Field Grammars

I define here (submodular) field grammars by combining the image grammars defined above with (submodular) MRFs. I do this by defining for each production $v : X \to Y\,Z$ an associated MRF over the full image

$$E_v(\mathbf{y}^v, \mathcal{I}) = \sum_{p \in \mathcal{I}} \theta_p^v(y_p^v) + \sum_{(p,q) \in \mathcal{I}} \theta_{pq}^v(y_p^v, y_q^v). \tag{5.3}$$

A copy of this MRF is instantiated each time an instance (equivalently, a token, as this relates to the well-known type-token distinction) of X is parsed as v, in the same way that each instance of a symbol uses the same categorical distribution to select productions. In particular, an instance of a symbol has an associated region $\mathcal{R} \subseteq \mathcal{I}$ and the MRF instantiation for this instance is simply the subset of the full-image MRF that contains

all of the nodes in $\mathcal{R}$ and all of the edges between the nodes in $\mathcal{R}$. The energy of this instance is thus $E_v(\mathbf{y}^v, \mathcal{R}) = \sum_{p \in \mathcal{R}} \theta_p^v(y_p^v) + \sum_{(p,q) \in \mathcal{R}} \theta_{pq}^v(y_p^v, y_q^v)$, the labeling distribution is $P(\mathbf{y}^v | v, \mathcal{R}) = \exp(-E_v(\mathbf{y}^v, \mathcal{R}))$, and the energy of a parse tree is

$$E(t, \mathcal{I}) = \sum_{(v,\mathcal{R}) \in t} w_v + E_v(\mathbf{y}^v, \mathcal{R}), \tag{5.4}$$

where the weights $\{w_v\}$ parameterize each symbol's categorical distribution over productions and the probability of a parse tree is $P(t, \mathcal{I}) \propto \exp(-E(t, \mathcal{I}))$. Each node $(v, \mathcal{R})$ in the parse tree contains an instance of its production v. To simplify notation, I will omit $v, \mathcal{I},$ and $\mathcal{R}$ when clear from context and sum over just v. I refer to this model as a field grammar $G = (N, \Sigma, R, S, \Theta)$ parameterized by Θ, which contains both the categorical weights and the MRF parameters.

The MRFs in a field grammar can be parameterized arbitrarily but, in order to permit efficient MAP inference, each term θ_{pq}^v must satisfy the previously-stated binary submodularity condition for all edges (p, q) and all productions $v : X \to Y_1 Y_2$ once the grammar has been converted to one in which each production has only two subconstituents, which is always possible and in the worst case increases the grammar size quadratically [68]. Note that it is easy to extend this to the non-binary case by instead requiring that the pairwise terms satisfy the α-expansion or $\alpha\beta$-swap conditions [21], for example, but I focus on the binary case here for simplicity. It is also necessary for every production $v \in R$, and for every production c that is a descendant of v in the grammar, that $\theta_{pq}^v(y_p^v, y_q^v) \geq \theta_{pq}^c(y_p^c, y_q^c)$ for all possible labelings $(y_p^v, y_q^v, y_p^c, y_q^c)$, where $y_p^v, y_q^v \in \mathcal{Y}_v$ and $y_p^c, y_q^c \in \mathcal{Y}_c$, to ensure that segmentations of higher-level productions are submodular relative to their descendants. This assumption captures a natural property of composition in images: that objects have larger regions than their parts. This means that the ratio of boundary length to region area is smaller for a symbol relative to its descendants, and thus its pairwise terms should be stronger. A grammar that satisfies these conditions is a *submodular field grammar* (SFG). Figure 5.1 shows a partial example of a (submodular) field grammar applied to scene

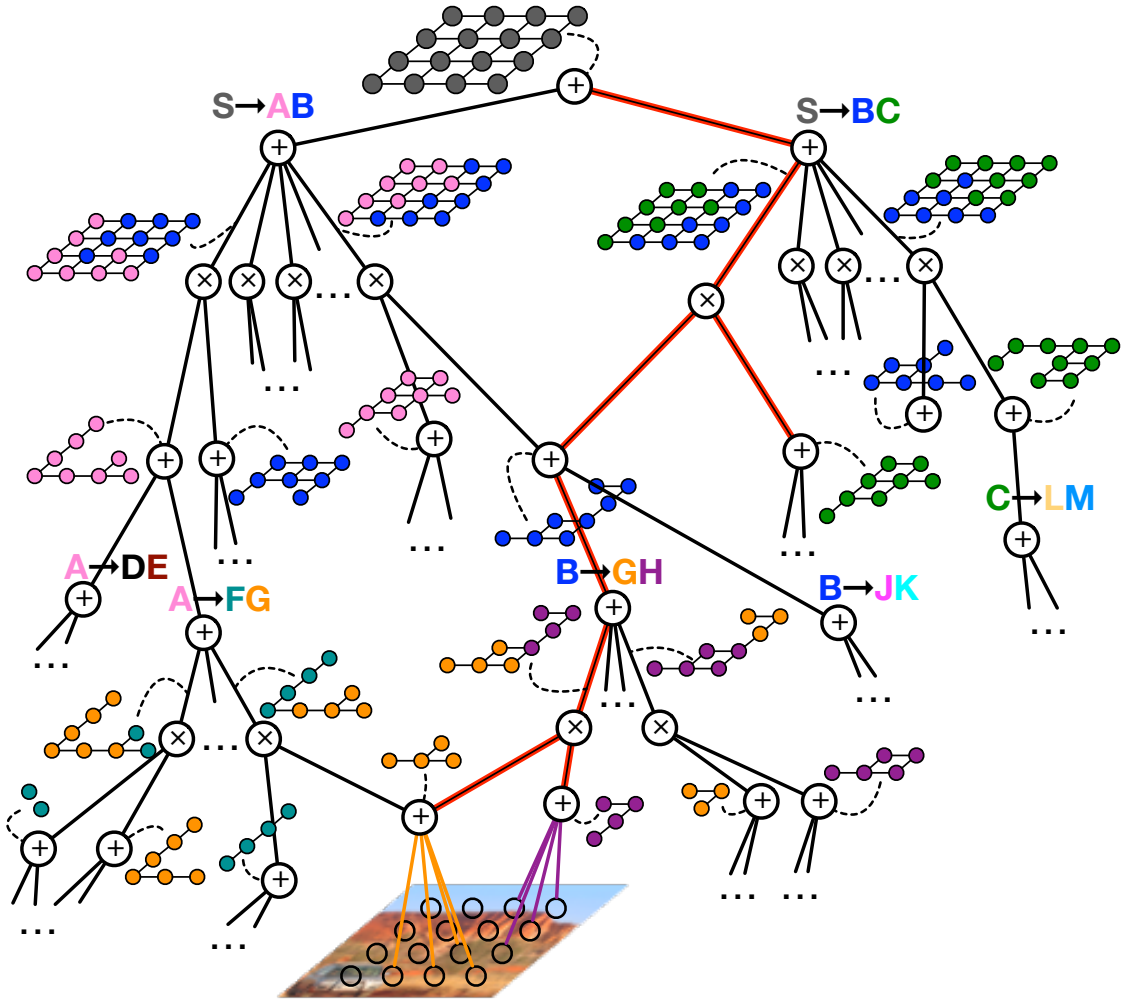


Figure 5.1: A DAG representing some of the possible production and labeling choices when parsing an image with an SFG. Each sum node represents either a choice of productions or a choice of labelings. Product nodes denote the partition of a region defined by its labeling, where an MRF node's color denotes its particular label for the current production. Red edges denote a (partial) parse tree for the image shown at the bottom. Best viewed in color.

understanding, demonstrating the interleaved choices of productions and labelings, and the subregion decompositions resulting from these choices.

5.3.1 Relationship to Other Models

Planar MRFs

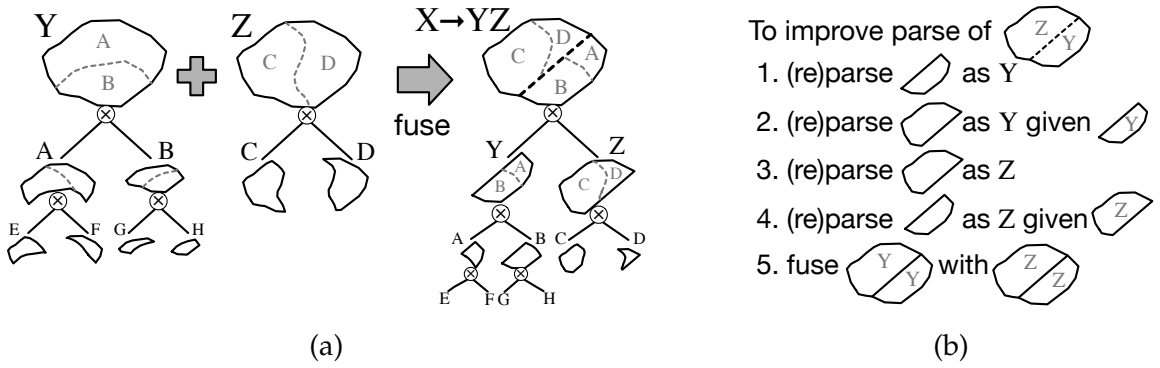
While I have introduced SFGs as image grammars with MRFs at each production, SFGs can also be described as flat MRFs with labels defined by a grammar. In particular, an SFG defines a planar MRF with one label for each path in the corresponding grammar, where the number of such paths is exponential in the height of the grammar. This can be seen as follows. Recall that a parse tree over some region $\mathcal{R}$ has energy $E(t, \mathcal{R}) = \sum_{v \in t} w_v + E_v(\mathbf{y}^v, \mathcal{R}_v)$. This energy can be rewritten as

$$E(t, \mathcal{R}) = w(t) + \sum_{p \in \mathcal{R}} \theta^t_p + \sum_{(p,q) \in \mathcal{R}} \theta^t_{pq}, \tag{5.5}$$

where $w(t) = \sum_{v \in t} w_v$; $[\cdot]$ is the indicator function; $\theta^t_p = \sum_{v \in t} \theta^v_p(y^v_p) \cdot [p \in \mathcal{R}_v]$; and $\theta^t_{pq} = \sum_{v \in t} \theta^v_{pq}(y^v_p, y^v_q) \cdot [(p, q) \in \mathcal{R}_v]$. This now describes a flat MRF in which θ^t_p and θ^t_{pq} are the unary and pairwise terms. Note that inference in this flat MRF is not any easier, and may in fact be harder, because it ahs an exponentially-large set of labels (one per grammar path) and the hard constraints of the grammar must now be enforced explicitly (e.g., that each instance of a symbol can only be produced once). Further, the benefit of the grammar-based formulation is that it easily enables sub-parse reuse, which can result in exponential reductions in inference complexity while also improving sample complexity. For example, consider reusing a Wheel symbol among many vehicle types. Instead of having to learn and perform inference for each Wheel symbol (once per vehicle type and per vehicle-parent type, etc.), only one Wheel need be learned and inference on it performed only once.

Sum-Product Networks.

Beyond PCFGs and MRFs, SFGs also extend sum-product networks (SPNs) [48, 114]. An SFG defines an SPN containing a sum node for each possible region of each nonterminal, a



(a) (b)

Figure 5.2: The two main components of SFG-PARSE: (a) Parsing a region $\mathcal{R}$ as $X \to YZ$ by fusing two parses of $\mathcal{R}$ as $Y \to AB$ and as $Z \to CD$, and (b) Improving the parse of $\mathcal{R}$ as $X \to YZ$ by (re)parsing each of its subregions, taking the union of the new Y and Z parses of $\mathcal{R}$, and then fusing these new parses. See text for more details

product node for each segmentation of each production of each possible region of each nonterminal, and a leaf function on the pixels of the image for each possible region of the image for each terminal symbol. The children of the sum node s for nonterminal X_s with region $\mathcal{R}_s$ are all product nodes r with a production $v_r : X_s \to Y_1 \dots Y_k$ and region $\mathcal{R}_{v_r} = \mathcal{R}_s$. Each product node corresponds to a labeling $\mathbf{y}^{v_r}$ of $\mathcal{R}_{v_r}$ and the edge to its parent sum node has weight $\exp(-w_v - E(\mathbf{y}^{v_r}, \mathcal{R}_{v_r}))$. The children of product node r are the sum or leaf nodes with matching regions that correspond to the constituent nonterminals or terminals of v_r, respectively. Figure 5.1 shows a partial mapping of an SFG to an SPN. Note that this underlying SPN is decomposable, because the scopes of the children of each product node form a partition of their parent's scope, but not complete (aka. smooth). However, the sum-product theorem shows that completeness is not a necessary condition for tractable inference and that no corrective factor is necessary when operating in the min-sum semiring, which is what is used for finding the approximate optimal parse of an SFG.

A key benefit of SFGs in comparison to previous grammar-based approaches is that regions can have arbitrary shapes and are not restricted to a small class of shapes such as

rectangles [114, 154]. This flexibility is important when parsing images, as real-world objects and abstractions can take any shape, but it comes with a combinatorial explosion of possible parses. Thus, the SPN resulting from an SFG must explicitly represent each possible subregion of the image and is necessarily exponentially large, meaning that inference in this SPN is intractable. This is the same reason that PCFGs are intractable when used to model images. Despite the intractability of SPNs for scene understanding, the approximate MAP parse of an image with respect to an SFG can be computed efficiently by exploiting submodularity, which I show in the following section. This enables efficient parsing of images into a hierarchy of arbitrarily-shaped regions and objects, yielding a very expressive model class.

5.4 Inference in SFGs

When trying to understand natural scenes, it is important to recognize and reason about the high-level relationships between objects. These relationships can be identified by finding the MAP parse of an image with respect to a grammar that encodes these structures, such as a submodular field grammar. The flat semantic labels traditionally used in scene understanding can also be recovered from this parse if they are encoded in the grammar, e.g., as the terminal symbols.

For natural language, the optimal parse of a PCFG can be recovered exactly in time cubic in the length of the sentence with the CYK algorithm [62], which uses dynamic programming to efficiently parse a sentence in a bottom-up pass through the grammar. This is possible because each sentence only has a linear number of split points, meaning that all sub-spans of a sentence can be efficiently represented and enumerated. The key operation in the CYK algorithm is to compute the optimal parse of a given span s (i.e., contiguous sub-sentence) as a given production $v : X \to YZ$ by explicitly iterating over all split points i of that span, computing the probability of parsing s as production v with split i, and choosing the split point with highest probability. The probability of parsing s as v with split i is defined recursively as the product of $P(v|\text{head}(v))$ and the respective probabilities of the optimal parses of the two sub-spans as Y and Z, respectively. CYK

uses dynamic programming to cache the optimal parse of each sub-span as each symbol to avoid re-computing these unnecessarily.

Unfortunately, a direct application of CYK to images is intractable because it is infeasible to enumerate all subregions of an image. Instead, I propose to construct (and cache) a parse of the entire image for each production and then use subsets of this parse to define the parse of each subregion, similar to how distributions over subregions are defined in SFGs. As in CYK, the optimal parse of a region as a particular production is defined recursively, but it is intractable to explicitly enumerate all possible splits (segmentations). Instead, inference must exploit submodularity to find the locally optimal parse from an exponentially large set without enumerating all such parses. I refer to this process of optimally combining the parses of the subconstituents of a production as *fusion*, as it is analogous to the fusion moves of Lempitsky et al. [92].

5.4.1 Parse Tree Construction by Fusion

Following Lempitsky et al. [92], let $\mathbf{y}^0, \mathbf{y}^1 \in \mathcal{Y}^n$ be two labelings of a submodular MRF with energy $E(\mathbf{y}) = \sum_p \theta_p(y_p) + \sum_{pq} \theta_{pq}(y_p, y_q)$ and let $C(\mathbf{y}^0, \mathbf{y}^1) = \{\mathbf{y}^c\}$ denote the set of combinations of $\mathbf{y}^0$ and $\mathbf{y}^1$. A labeling $\mathbf{y}^c = (y_0^c, \ldots, y_n^c)$ is a combination of $\mathbf{y}^0 = (y_0^0, \ldots, y_n^0)$ and $\mathbf{y}^1 = (y_0^1, \ldots, y_n^1)$ iff each label in $\mathbf{y}^c$ is taken either from $\mathbf{y}^0$ or $\mathbf{y}^1$ such that $y_p^c \in \{y_p^0, y_p^1\}$ for all pixels $p = 1 \ldots n$. The fusion $\mathbf{y}^*$ of $\mathbf{y}^0$ and $\mathbf{y}^1$ is then defined as the minimum energy combination $\mathbf{y}^* = \arg\min_{\mathbf{y} \in C(\mathbf{y}^0, \mathbf{y}^1)} E(\mathbf{y})$. Under certain conditions on E, fusing two labelings is a submodular minimization.

Recall from Section 5.3.1 that each parse tree t of an SFG can equivalently be thought of as a particular labeling of a planar MRF having one label per path in the grammar. With a slight abuse of notation, I use t to represent both the full parse tree and the corresponding labeling in the planar MRF. Now, let $v : X \to Y_1 Y_2$ be a production and t_1, t_2 be parses of some region $\mathcal{R} \subseteq \mathcal{I}$ as productions $u_1 : Y_1 \to Z_1 Z_2$ and $u_2 : Y_2 \to Z_3 Z_4$, respectively, where a parse of a region as a production u is simply a parse of that region by the subset of the grammar that contains only the symbols and productions that can be produced by u. Because t_1 and t_2

are MRF labelings, they can be fused to create a new parse tree t_v in which each pixel in $\mathcal{R}$ is labeled with the same path that it had in either t_1 or t_2. When this occurs, v is prepended to each pixel's label, which is equivalent to adding $(v, \mathcal{R})$ as the new root node of t_v.

Definition 5.1. *For production $v : X \to Y_1 Y_2$ and parse trees t_1, t_2 over region $\mathcal{R}$ with head symbols Y_1, Y_2, the* fusion *of t_1 and t_2 as v is the minimum energy parse tree $t_v = \arg\min_{t \in C(t_1, t_2)} E(t, \mathcal{R})$ constructed from the combination of t_1 and t_2, with $(v, \mathcal{R})$ appended as the root.*

Figure 5.2a shows an example of fusing two parse trees to create a new parse tree. Although fusion requires finding the optimal labeling from an exponentially large set, the fusion energy function is submodular for SFGs and thus two parse trees can be efficiently fused with a single graph cut.

Proposition 5.1. *The fusion of parse trees t_1, t_2 over region $\mathcal{R}$ with head symbols Y_1, Y_2 for a production $v : X \to Y_1 Y_2$ is a submodular minimization.*

Proof. First, let every submodular MRF energy $E^u(\mathbf{y}^u, \mathcal{R})$ be in normal form such that $\theta^u_{pq}(y, y) = 0$ and $\theta^u_{pq}(y^u_p, y^u_q) \geq 0$ for all labels $y, y^u_p, y^u_q \in \mathcal{Y}_u$, where any submodular energy can be reparameterized into normal form in time linear in the region size [75].

Now, the fusion of parse trees t_0, t_1 is given by $t^* = \arg\min_{t \in C(t_0, t_1)} E(t, \mathcal{R})$, where $E(t, \mathcal{R}) = \sum_{v \in t} w(t) + \sum_{p \in \mathcal{R}} \theta^t_p + \sum_{(p,q) \in \mathcal{R}} \theta^t_{pq}$ and $\theta^t_{pq} = \sum_{v \in t} \theta^v_{pq}(y^v_p, y^v_q)$. However, because each θ^v_{pq} is in normal form and any parse tree can contain only one production $c \in t$ in which $y^c_p \neq y^c_q$ for any neighboring pixels p, q (because these pixels are placed in different regions in all descendants of the region in which they are labeled differently), it follows that the summation for θ^t_{pq} contains at most one non-zero term and thus $\theta^t_{pq} = \theta^c_{pq}(y^v_p, y^v_q)$.

Finally, by introducing an auxiliary binary variable $\mathbf{z} \in \{0, 1\}^{|\mathcal{R}|}$, the combination of t_0 and t_1 can be defined as $t^c(\mathbf{z}) = t_0 \circ (1 - \mathbf{z}) + t_1 \circ (\mathbf{z})$, where $\circ$ is the Hadamard (element-wise) product. The fusion problem can now be written as a binary minimization problem $t^* = \arg\min_{\mathbf{z} \in \{0,1\}^{|\mathcal{R}|}} E(t(\mathbf{z}), \mathcal{R})$, where the energy is submodular if $\theta^{t(z_p=0, z_q=1)}_{pq} + \theta^{t(z_p=1, z_q=0)}_{pq} \geq \theta^{t_0}_{pq} + \theta^{t_1}_{pq}$. Because the pairwise terms are in normal form it follows that

$\theta_{pq}^{t_0} = \theta_{pq}^{c_0}$ and $\theta_{pq}^{t_0} = \theta_{pq}^{c_1}$, where c_0 and c_1 are the productions in t_0 and t_1, respectively, at which pixels p and q were labeled differently (note that the values of their labels are unimportant here). Further, $\theta_{pq}^{t(z_p=0,z_q=1)} = \theta_{pq}^{v}(Y_0, Y_1)$ and $\theta_{pq}^{t(z_p=1,z_q=0)} = \theta_{pq}^{v}(Y_1, Y_0)$ because by choosing them from different trees, the production v that is being fused is now the production at which these pixels are first labeled differently. However, since (from the definition of an SFG) $\theta_{pq}^{v}(y_p^v, y_q^v) \geq \theta_{pq}^{c}(y_p^c, y_q^c)$ for c any possible descendant production of v and for all labelings, then it follows that $\theta_{pq}^{v}(Y_0, Y_1) \geq \theta_{pq}^{c_0}$ and $\theta_{pq}^{v}(Y_1, Y_0) \geq \theta_{pq}^{c_1}$ and thus the submodularity condition holds. □

Finally, I define the union of two parse trees $t = t_1 \cup t_2$ that contain the same productions but are over disjoint regions (i.e., $\mathcal{R}_1 \cap \mathcal{R}_2 = \varnothing$) as the parse tree t in which the region of each node in t is the union of the regions of the corresponding nodes in t_1 and t_2.

5.4.2 SFG-Parse

Pseudocode for my novel parsing algorithm, SFG-Parse, is presented in Algorithm 5. SFG-Parse is an iterative move-making algorithm that efficiently and provably converges to a local minimum of the energy function. Currently, SFG-Parse applies only to non-recursive grammars, but I believe it will be straightforward to extend to full grammars.

SFG-Parse starts at the terminal symbols and moves backwards through the productions towards the start symbol (line 13), constructing and caching a parse of the full image as each production. The parse for each production is constructed by fusing the cached parses of that production's subconstituents (lines 21 and 22). In CYK, each span of a sentence is explicitly parsed as each production, making it straightforward to have multiple instances of a symbol, each with a different sub-parse. However, when parsing an SFG, each parse tree must remain internally consistent (i.e., the same symbol cannot be produced multiple times) to ensure that fusion is submodular. Thus, if only one parse tree were used per production, each instance of a symbol would have to be parsed with the exact same set of productions.

Algorithm 5 Compute the (approximate) MAP parse of an image with respect to a submodular field grammar.

Input: The image $\mathcal{I}$, a non-recursive SFG $G = (N, \Sigma, R, S, \Theta)$, and an (optional) input parse $\hat{t}$.
Output: A parse of the image, t^*, with energy $E(t^*, \mathcal{I}) \leq E(\hat{t}, \mathcal{I})$.

function SFG-PARSE($\mathcal{I}, G, \hat{t}$)
  // initialize terminal parses
  for each terminal $T \in \Sigma$ **do**
    $t_{\mathcal{R}_T} \leftarrow$ the trivial parse with all pixels parsed as T
  while the energy of any production of the start symbol S has not converged **do**
    // record each instance of each symbol that appears in $\hat{t}$
    for each node in $\hat{t}$ with production $u : X \to YZ$, region $\mathcal{R}_X$, and subregions $\mathcal{R}_Y, \mathcal{R}_Z$ **do**
      append $\mathcal{R}_Y, \mathcal{R}_Z$ to region lists $\mathcal{R}[Y], \mathcal{R}[Z]$ and set as the child regions of u for $\mathcal{R}_X$
    // initialize each instance-less symbol to have the entire image as its sole region
    for each symbol $X \in N$ with no regions in $\mathcal{R}[X]$ **do**
      add $\mathcal{R}_X = \mathcal{I}$ to $\mathcal{R}[X]$
    // perform an upward pass through the grammar to parse the SFG
    for each symbol $X \in N$, in reverse topological order **do**
      for each region $\mathcal{R}_X$ in region list $\mathcal{R}[X]$ **do** *// each region corresponds to an instance of* X
        for each production $v : X \to YZ$ of symbol X **do**
          // get a parse for each constituent of production v *and fuse these*
          if the child regions of production v for region $\mathcal{R}_X$ exist **then**
            $\mathcal{R}_Y, \mathcal{R}_Z \leftarrow$ the child regions of production v for region $\mathcal{R}_X$
          else
            $\mathcal{R}_Y, \mathcal{R}_Z \leftarrow$ heuristically choose instances of Y and Z
          $t_v, e_v \leftarrow$ fuse $t_{\mathcal{R}_Y}$ and $t_{\mathcal{R}_Z}$ as production v over region $\mathcal{R}_X$
          $t_{\bar{v}}, e_{\bar{v}} \leftarrow$ fuse $t_{\mathcal{R}_Y}$ and $t_{\mathcal{R}_Z}$ as production v over region $\mathcal{R}_{\overline{X}} = \mathcal{I} \backslash \mathcal{R}_X$ given t_v
        // choose best parse of $\mathcal{R}_X$
        $t_{\mathcal{R}_X}, e_{\mathcal{R}_X} \leftarrow$ the full parse $t_v \cup t_{\bar{v}}$ with lowest energy e_v
    // S only ever has a single region, which contains all of the pixels
    $\hat{t}, \hat{e} \leftarrow t_{\mathcal{R}_S}, e_{\mathcal{R}_S}$
  return $\hat{t}, \hat{e}$

To avoid this, SFG-PARSE permits multiple instances of a symbol X, one per unique path from the root to a production of X in $\hat{t}$, where $\hat{t}$ is the best parse from the previous iteration. This allows the number of instances of each symbol to grow or shrink at each

iteration. Detection of instances and their corresponding regions occurs on line 8. When SFG-PARSE detects an instance x of symbol X in $\hat{t}$ for a production $v : X \to YZ$, it also records pointers to its child instances y and z, which later determine the instances of Y and Z to fuse to create a parse of v (line 18).

If a symbol has no instances, either because it doesn't appear in $\hat{t}$ or because $\hat{t}$ was not provided, then that symbol is assigned the region containing the entire image as an instance (line 11), which serves as a powerful initialization method. If a symbol has no instances, then it did not appear in $\hat{t}$ so its parse can be constructed by fusing any instance of each of its production's subconstituents without affecting convergence. Since most symbols do not appear in $\hat{t}$, they only have a single instance (corresponding to the entire image) and the parse of that instance is used each time this symbol is fused. In the rare case that a symbol has multiple instances, one instance can be chosen either by estimating a bound on its energy or even randomly (line 20).

If a symbol X does have an instance in $\hat{t}$, SFG-PARSE first constructs a parse t_v of that instance's region $\mathcal{R}_X$ (line 21) and then constructs a parse of the remainder of the image $\mathcal{R}_{\overline{X}}$ as v given the partial parse t_v (line 22). The union of these two parses is a full parse of the entire image as v for this instance. Parsing an instance in two parts is necessary to ensure that SFG-PARSE never returns a worse parse. When constructing a parse of a region $\mathcal{R}_{\overline{X}}$ given a parse t_v of another (disjoint) region $\mathcal{R}_X$, the pairwise terms that span the two regions are included in the fusion energy function and the label of the pixel in $\mathcal{R}_X$ is set to its value in t_v (i.e., like conditioning in a probabilistic model). Figure 5.2b shows an inefficient version of the process of re-parsing an instance of X, where first the subregions labeled as Y and Z are re-parsed (steps 1-2), then the remaining pixels are re-parsed given the other parses (steps 3-4), and finally the unions of these parses are fused to get a parse of the region as X (step 5). However, for efficiency reasons, SFG-PARSE does not actually re-parse Y and Z for each production that produces them; instead, their parses are cached and re-used.

Finally, the parse of the production u that has the lowest energy over $\mathcal{R}_X$ is chosen as

the parse of $\mathcal{R}_X$ (line 24). At the end of the upward pass, the parse of the full image $\hat{t}$ is simply the parse of the start symbol's single region, which always contains all pixels (line 26).

5.4.3 Analysis of SFG-Parse

As shown in Theorem 5.1, SFG-Parse always converges to a local minimum of the energy function. Like other move-making algorithms, SFG-Parse explores an exponentially large set of moves at each step, so the returned local minimum is in general much better than the local minima returned by more local procedures [21]. Further, I typically observe convergence in fewer than ten iterations, with the majority of the energy improvement occurring in the first iteration. In order to prove Theorem 5.1, I first show that the parse of a subregion can be improved without making the overall parse energy worse.

Lemma 5.1. *Given a labeling $\mathbf{y}^v$ which fuses parse trees t_1, t_2 into t with root production v, energy $E(t, \mathcal{R})$, and subtree regions $\mathcal{R}_1 \cap \mathcal{R}_2 = \varnothing$ defined by $\mathbf{y}^v$, then any improvement Δ in $E(t_1, \mathcal{R}_1)$ also improves $E(t, \mathcal{R})$ by at least Δ, regardless of any change in $E(t_1, \mathcal{R} \backslash \mathcal{R}_1)$.*

Proof. Since the optimal fusion can be found exactly, and the energy of the current labeling $\mathbf{y}^v$ has improved by Δ, the optimal fusion will have improved by at least Δ. □

Convergence of SFG-Parse now follows from Lemma 5.1.

Theorem 5.1. *Given a parse $\hat{t}$ of S over the entire image with energy $E(\hat{t})$, each iteration of* SFG-Parse *constructs a parse t of S over the entire image with energy $E(t) \leq E(\hat{t})$, and since the minimum energy of an image parse is finite* SFG-Parse *will always converge.*

Proof. I will prove by induction that for all nodes $n \in \hat{t}$ with corresponding production $v : X \to YZ$, region $\mathcal{R}_X$, subtree $\hat{t}_{\mathcal{R}_X}$ over region $\mathcal{R}_X$, and child subtrees $\hat{t}_{\mathcal{R}_Y}, \hat{t}_{\mathcal{R}_Z}$ over regions $\mathcal{R}_Y, \mathcal{R}_Z$, that $E(t_{\mathcal{R}_X}) \leq E(\hat{t}_{\mathcal{R}_X})$ after one iteration. Since the start symbol S has only one region containing the entire image, this proves the claim.

Base case. Let $\hat{t}_{\mathcal{R}_X}$ be a subtree with region $\mathcal{R}_X$ and production $v : X \to Y$ containing

only a single terminal child and let $\{u_i = X \to Y_i\}$ be the set of productions of X (where such a $\hat{t}_{\mathcal{R}_X}$ must exist because the grammar is non-recursive and terminates). By definition, $t_v = \hat{t}_{\mathcal{R}_X}$, where t_v is the new parse of $\mathcal{R}_X$ as v, because terminal parses do not change for the same region. Then, since $t_{\mathcal{R}_X} = \arg\min_{u_i} E(t_{u_i})$ and $v \in \{u_i\}$, it immediately follows that $E(t_{\mathcal{R}_X}) \leq E(\hat{t}_{\mathcal{R}_X})$ and the claim holds.

Induction step. Let $n \in \hat{t}$ be a node in $\hat{t}$ with corresponding production $v : X \to YZ$, region $\mathcal{R}_X$, subtree $\hat{t}_{\mathcal{R}_X}$ over region $\mathcal{R}_X$, and child subtrees $\hat{t}_{\mathcal{R}_Y}, \hat{t}_{\mathcal{R}_Z}$ over regions $\mathcal{R}_Y, \mathcal{R}_Z$, such that $\mathcal{R}_Y \cup \mathcal{R}_Z = \mathcal{R}_X$ and $\mathcal{R}_Y \cap \mathcal{R}_Z = \varnothing$, and suppose that $E(t_{\mathcal{R}_Y}) \leq E(\hat{t}_{\mathcal{R}_Y})$ and $E(t_{\mathcal{R}_Z}) \leq E(\hat{t}_{\mathcal{R}_Z})$. From Lemma 1, it follows that the parse t_v computed from fusing $t_{\mathcal{R}_Y}$ and $t_{\mathcal{R}_Z}$ in $\mathcal{R}_X$ as v has energy $E(t_v) \leq E(\hat{t}_{\mathcal{R}_X})$ (since the fusion can always choose the same labeling as in $\hat{t}_{\mathcal{R}_Y}$). Then, since $t_{\mathcal{R}_X} = \arg\min_{u \in \{u_X : \text{head}(u_X) = X\}} E(t_u)$, where $\{u_X\}$ are the productions of X, it follows that $E(t_{\mathcal{R}_X}) \leq E(t_v)$ and thus $E(t_{\mathcal{R}_X}) \leq E(\hat{t}_{\mathcal{R}_X})$ and the claim follows. □

As shown in Proposition 5.2, each iteration of SFG-Parse has worst-case complexity $O(|G|c(n)n)$, where n is the number of pixels in the image and $c(n)$ is the complexity of the underlying graph cut algorithm used, which is worst-case low-order polynomial, but nearly linear-time in practice [20, 21]. The additional factor of n is due to the number of regions (i.e., instances) of each symbol, which in the worst case can be $O(n)$ but in practice is almost always a small constant (often one). Thus, SFG-Parse typically runs in time $O(|G|c(n))$.

Proposition 5.2. *Let $c(n)$ be the time complexity of computing a graph cut on n pixels and $|G|$ be the size of the grammar defining the SFG. Then each iteration of* SFG-Parse *takes time $O(|G|c(n)n)$.*

Proof. Let k be the number of productions per nonterminal symbol and N be the nonterminals. The three main loops of the algorithm have complexity $|N|$, n (because there can be at most n regions and the regions are disjoint), and k, respectively. For line 18, the choice of parses for productions in $\hat{t}$ takes constant time, and the parses for productions that are not in $\hat{t}$ (line 20) can be chosen arbitrarily and thus also in constant time. For lines 21-22, the fusion of a region $\mathcal{R}$ has complexity $O(|\mathcal{R}|+c(|\mathcal{R}|)) = O(c(|\mathcal{R}|))$, so the

worst-case complexity of the inner loop occurs when $\mathcal{R}$ is empty or is the full image, giving complexity $O(c(n))$. Thus, the total complexity of each iteration of SFG-Parse is $O(|N|k \cdot c(n) \cdot n) = O(|G|c(n)n)$. □

Note that a straightforward application of α-expansion to image parsing that uses one label for every possible parse in the grammar requires an exponential number of labels in general, and thus has exponential time complexity.

SFG-Parse can be extended to productions with more than two subconstituents by replacing the internal graph cut used to fuse subtrees with a multi-label algorithm such as α-expansion. SFG-Parse would still converge because each subtree would still never decrease in energy. An algorithm such as QPBO [75] could also be used, which would obviate the submodularity requirement.

5.5 SFG Learning

An SFG is parameterized by its production costs $\mathbf{w}_\mathrm{s} = \{w_v : v \in R\}$, which are the log-space version of the SPN (or grammar) weights, and its MRF weights $\mathbf{w}_\mathrm{m}$. The parameters of an SFG can be learned in a multitude of ways, but I propose here an approach that follows directly from SFGs being a fusion of grammars and MRFs. In particular, given the regions of each symbol, an SFG reduces to a stochastic grammar. Conversely, given the productions, an SFG reduces to an MRF. I thus propose to learn SFG parameters using an alternating-minimization-style algorithm, where the grammar weights $\mathbf{w}_\mathrm{s}$ are first updated while the MRF parameters held fixed and then the MRF weights $\mathbf{w}_\mathrm{m}$ are updated while the grammar weights held fixed, with this process iterating until convergence. While it is possible to learn $\mathbf{w}_\mathrm{s}$ and $\mathbf{w}_\mathrm{m}$ simultaneously, my preliminary investigations indicate that learning them separately provides a more stable approach, where symbols are first associated with different image patches (or features) by updating $\mathbf{w}_\mathrm{s}$, and then each symbol's region and appearance weights $\mathbf{w}_\mathrm{m}$ are fit to that symbol's image patches (or features).

To update the weights, recall that the stochastic grammars underlying SFGs are also

SPNs, which enables the use of SPN-learning methods. In particular, both SPNs and MRFs can be learned both generatively and discriminatively, and SFGs are no different; I focus here on the discriminative case, but the generative case is similar. As with both SPNs and MRFs, the derivative of the conditional log-likelihood of an SFG with respect to a weight w_i is simply the difference of the expected count of the corresponding production (or pixel or edge) over all parse trees that are compatible with both the labels and the image and the expected count over parse trees compatible with only the image; i.e.,

$$\frac{\partial}{\partial w_i} \log P_{\mathbf{w}}(\mathbf{y}|\mathcal{I}) = \mathbb{E}_{t \in \mathcal{T}_{\mathbf{w}}(\mathbf{y},\mathcal{I})}[n_i(t) \mid \mathbf{y}, \mathcal{I}] - \mathbb{E}_{t' \in \mathcal{T}_{\mathbf{w}}(\mathcal{I})}[n_i(t') \mid \mathcal{I}], \tag{5.6}$$

where $\mathbf{y}$ are the query variables, $\mathcal{T}_{\mathbf{w}}(\mathbf{y}, \mathcal{I})$ and $\mathcal{T}_{\mathbf{w}}(\mathcal{I})$ are the sets of parse trees compatible with their respective arguments, and $n_i(t)$ is the count of weight w_i in t. Unfortunately, since no datasets of parsed images exist to train on, both expectations are intractable. However, an effective approximation to the second expectation is to simply use the counts from the MAP parse, which is accurate if it has most of the probability mass; this is known as voted perceptron [30] and has been used to efficiently train both MRFs [135] and SPNs [47]. In SPNs, both expectations are tractable but are still replaced with their MAP state to overcome vanishing gradients. I propose to extend this method to SFGs, and approximate each expectation with the counts from its respective MAP parse, as found by SFG-Parse. The gradient update is then $\frac{\partial}{\partial w_i} \log P_{\mathbf{w}}(\mathbf{y}|\mathcal{I}) \approx n_i(t^*_{\mathbf{y}\mathcal{I}}) - n_i(t^*_{\mathcal{I}})$, where $t^*_{\cdot} = \arg\min_{t \in \mathcal{T}_{\mathbf{w}}(\cdot)} E(t, \mathcal{I})$. I leave SFG structure learning to future work, but am excited about combining SPN [2, 48] and arithmetic circuit [120] structure learning algorithms with methods for grammar induction [74, 113].

5.6 Empirical Evaluation of SFGs and SFG-Parse

I evaluated SFGs and SFG-Parse by parsing images from the Stanford background dataset (SBD) [50] under two different settings, both using features from DeepLab [28, 29], a state-of-the-art convolutional semantic segmentation approach. In the first experiment, I

evaluated the performance of SFG-PARSE by comparing its runtime and accuracy to that of α-expansion and max-product belief propagation. In the second experiment, I induced grammars from SBD images and compared the segmentation performance of SFGs on DeepLab features to that of a planar submodular MRF on those same features and to those features alone.

As discussed in Section 5.3, the energy of each segmentation of a region for a given production is defined by an MRF $E(\mathbf{y}^v, \mathcal{R}_v) = \sum_{p \in \mathcal{R}_v} \theta^v_p(y^v_p; \mathbf{w}) + \sum_{(p,q) \in \mathcal{R}_v} \theta^v_{pq}(y^v_p, y^v_q; \mathbf{w})$, parameterized by weights $\mathbf{w}$. The unary and pairwise terms in E can be defined arbitrarily, as long as the resulting energy is submodular. In my experiments, I defined the unary terms for terminals $T \in \Sigma$ as a linear function of the image features $\theta^v_p(y^v_p = T; \mathbf{w}) = \mathbf{w}_T^\top \phi^U_p$, where ϕ^U_p is a feature vector representing the local appearance of pixel p. Unary terms for nonterminals $X \in N$ were defined as $\theta^v_p(y^v_p = X; \mathbf{w}) = w^v_{pX}$, where w^v_{pX} is a (learnable) parameter that specifies how likely this pixel is to be labeled as X. This allowed each production to encode the regions of the image associated with each of its constituents.

The pairwise terms are also quite flexible, but in my experiments I used the standard contrast-dependent pairwise boundary potential (e.g., Shotton et al. [132]) defined for each production v and each pair of adjacent pixels p, q as

$$\theta^v_{pq}(y^v_p, y^v_q; \mathbf{w}) = w^{\text{BF}}_v \exp(-\beta^{-1} ||\phi^B_p - \phi^B_q||^2) \cdot [y^v_p \neq y^v_q], \tag{5.7}$$

where β is half the average image contrast between all adjacent pixels in an image, w^{BF}_v is the boundary factor that controls the relative cost of this term for each production, ϕ^B_p is the pairwise per-pixel feature vector, and $[\cdot]$ is the indicator function.

5.6.1 Inference Evaluation

To evaluate the performance of SFG-PARSE, I programmatically generated grammars while varying their height, number of productions per nonterminal, and strength of the pairwise (boundary) terms. The same grammar, DeepLab features, and randomly-selected images

were used to evaluate each algorithm. I compared SFG-Parse to α-expansion on a flat pairwise MRF containing a label for each possible parse path in the grammar and to max-product belief propagation (BP) on a multi-level (3-D) pairwise MRF with the same height as each grammar. Further details on these models and the MRF parameterization are provided below. Due to the flat encoding, α-expansion must iterate over an exponential number of labels; however, once it converges, its energy is within a constant factor of the global minimum [21] and is thus a good surrogate for the true global minimum, which is intractable to compute.

All experiments were run on the same computer, which was a dual 20-core 2.2 GHz Intel Xeon E5-2698 with 512 GB of RAM. Each algorithm was limited to a single thread.

Experiment: Increasing boundary strength

Increasing the boundary strength of an MRF makes inference more challenging, as individual pixel labels cannot be easily flipped without large side effects. To test the performance of SFG-Parse as a function of boundary strength, I constructed a grammar parameterized as above with 2 layers of nonterminals (not including the start symbol), each containing 3 nonterminal symbols with 4 binary productions to the next layer. I used a single weight w_{BF} to parameterize all pairwise (boundary) terms in the MRF of every production. Figure 5.3 plots the mean average pixel accuracy of the parses returned by each algorithm vs. w_{BF} (the x-axis is log-scale). SFG-Parse returns parses with almost identical accuracy (and energy) to α-expansion. BP also returns comparable accuracies, but almost always returns invalid parses with infinite energy (if it converges at all) that contain multiple productions of the same object or a production of a symbol Y even though the pixel is labeled as symbol X.

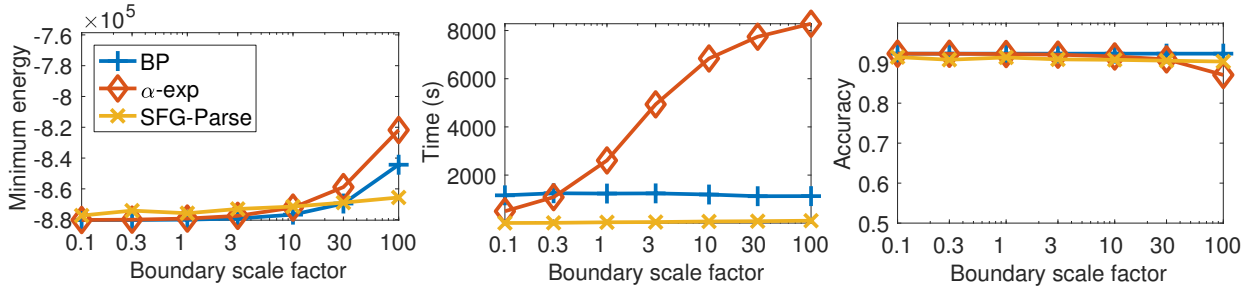


Figure 5.3: The (a) best energy, (b) total running time, and (c) resulting semantic segmentation accuracy (mean average pixel accuracy) for belief propagation, α-expansion, and SFG-Parse when varying boundary strength. Each data point is the average value over (the same) 10 images.

Experiment: Increasing grammar height

In general, the number of paths in a grammar is exponential in its height, so the height of the grammar controls the complexity of inference and thus the difficulty of parsing images. For this experiment, I set w_{BF} to 20 and constructed a grammar with four nonterminals per layer, each with three binary productions to the next layer. Figure 5.4 shows the effect of grammar height on (a) minimum energy found, (b) total inference time (to convergence or a maximum number of iterations, whichever first occurred), and (c) semantic segmentation accuracy (mean average precision) for each algorithm. As expected from Proposition 5.2, the time taken for SFG-Parse scaled linearly with the height of the grammar, which is within a constant factor of the size of the grammar when all other parameters are fixed. Conversely, inference time for both α-expansion and BP scaled exponentially with the height of the grammar. Again, the energies and corresponding accuracies achieved by SFG-Parse were nearly identical to those of α-expansion.

Experiment: Increasing number of productions per nonterminal

The number of paths in the grammar is also directly affected by the number of productions per symbol. For this experiment, I set w_{BF} to 20 and constructed a grammar with 2 layers of

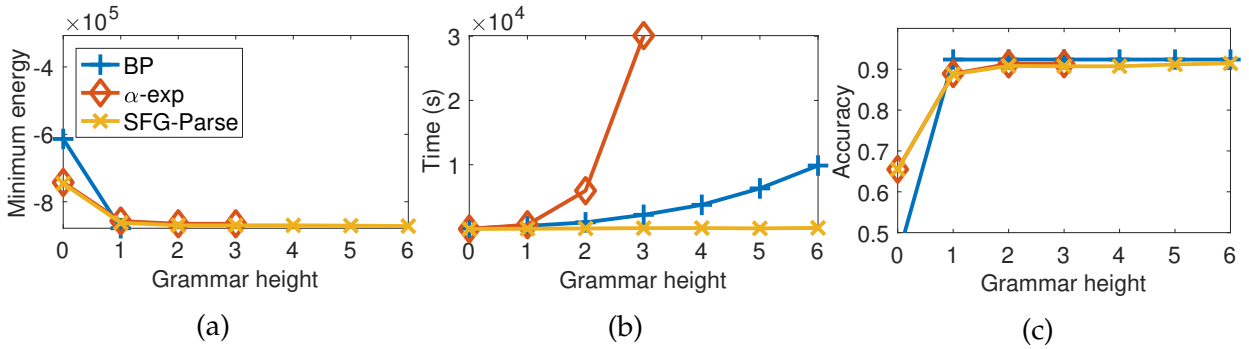


Figure 5.4: The (a) best energy, (b) total running time, and (c) resulting semantic segmentation accuracy (mean average pixel accuracy) for belief propagation, α-expansion, and SFG-Parse when varying grammar height. Each data point is the average value over (the same) 10 images. Missing data points for α-expansion indicate that it ran out of memory. Missing data points for BP indicate that it returned infinite energy (left). Low accuracies for grammar height 0 are a result of the grammar being insufficiently expressive.

nonterminals, each with 4 nonterminal symbols. Figure 5.5 shows the effect of increasing the number of productions per nonterminal, which again demonstrates that SFG-Parse is far more efficient than either α-expansion or BP as the complexity of the grammar increases, but still finds comparable solutions.

α-expansion and 3-D MRF Details

I compared SFG-Parse to running α-expansion on a flat pairwise MRF and to max-product belief propagation over a multi-level (3-D) pairwise grid MRF. Each label of the flat MRF corresponds to a possible path in the grammar from the start symbol to a production to one of its constituent symbols, etc, until reaching a terminal. In general, the number of such paths is exponential in the height of the grammar. The unary term for a label in the flat MRF are the sum of the unary terms along the corresponding path and the pairwise term for a pair of labels is the pairwise term of the first production at which their constituents differ. For any two labels with paths that choose a different production of the same symbol (and have the same path from the start symbol) I assign infinite cost to enforce the restriction

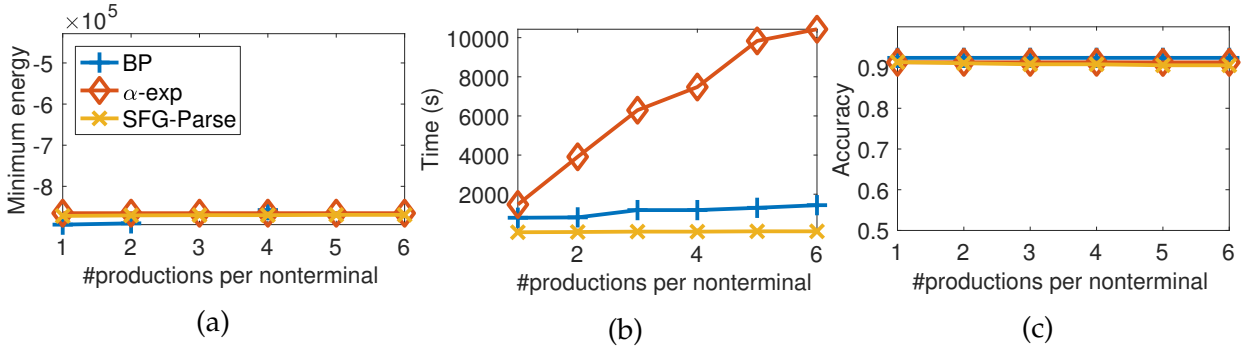


Figure 5.5: The (a) best energy, (b) total running time, and (c) resulting semantic segmentation accuracy (mean average pixel accuracy) for belief propagation, α-expansion, and SFG-Parse when varying grammar height. Each data point is the average value over (the same) 10 images. Missing data points for BP indicate that it returned infinite energy (left).

that an object can only have a single production of it into constituents.

The multi-layer MRF is constructed similarly. The number of levels in the MRF is equal to the height of the DAG corresponding to the grammar used. The labels at a particular level of the MRF are all (production, constituent) pairs that can occur at that level in the grammar. The pairwise term between the same pixel in two levels is 0 when the parent label's constituent equals the child label's production head, and ∞ otherwise. Pairwise terms within a layer are defined as in the flat MRF with infinite cost for incompatible labels (i.e., two neighboring pixels parsed as different productions of the same symbol), unless two instances of that nonterminal could be produced at that level by the grammar.

5.6.2 Model Evaluation

In order to evaluate whether natural scenes exhibit the part-subpart structure over arbitrarily-shaped regions that SFGs can capture, but that existing methods cannot, I induced grammars on SBD images and computed the mean pixel accuracy of the terminal labeling (i.e., the per-pixel semantic labels) from the parse tree returned by SFG-Parse.

I first over-segmented each of 143 images at 4 different levels of granularity using

the method and code of Isola et al. [66] and intersected the most fine-grained of these over-segmentations with the regions of the true labels. I then created a unique grammar for each image by taking that image's over-segmentations and the over-segmentations of four other randomly chosen images and adding a symbol for each contiguous region. To these symbols, I added a production between each set of overlapping segments for each subsequent pair of granularity levels within each image. I then generated three productions for each symbol by (for each production) randomly selecting four overlapping regions from the next level of segmentation granularity across all images. Finally, I added productions from the symbols in the most granular level to the terminal symbols, where each terminal production can produce only those labels that occur in the region of its head symbol. To make the comparison fair, I also restricted the labels that can be produced by other models in the same way.

On average, each induced grammar had 860 symbols and 1250 productions with 5 constituents each. The features output by DeepLab (from the layer preceding the softmax in the DeepLab architecture) were used as the per-pixel unary costs in the MRFs of the terminal productions. All productions had uniform probability and the same MRF parameters were used across all images to ensure that any improvement in performance is due solely to the structure of the underlying grammar, as no learning was used. For the MRF weights, I set w_v^{BF} to 5 for all productions and all edges, but used the contrast-dependent pairwise boundary potential defined above for the pairwise terms.

DeepLab	DeepLab + MRF	DeepLab + SFG
87.77	87.93	**90.03**

Table 5.1: The mean pixel accuracy on 143 SBD images when parsed with DeepLab, DeepLab with an MRF on the output features, and DeepLab with an SFG on the output features.

After parsing each image with respect to its grammar, I computed the mean pixel accuracy of the terminal labeling of the parse. I compared this to the accuracy of the

DeepLab features alone and to the accuracy of a standard flat submodular MRF over the DeepLab features, with pairwise terms set in the same way as in the SFGs. These results are shown in Table 5.1, which shows a 20% relative decrease in error for SFGs, which is quite remarkable given how well the DeepLab features do on their own and how little the flat MRF helps. This indicates that natural scenes do exhibit high-level compositional structure and that SFGs are able to exploit this structure to improve scene understanding.

5.7 Conclusion

This chapter proposed submodular field grammars (SFGs), a novel type of stochastic image grammar that combines the expressivity of image grammars with the efficient combinatorial optimization capabilities of submodular Markov random fields. SFGs are the first image grammar to enable efficient parsing of objects with arbitrary region shapes. To achieve this, I proposed SFG-Parse, a move-making algorithm that exploits submodularity in order to find the approximate MAP parse of an SFG. Analytically, I showed that SFG-Parse is both convergent and efficient, since each iteration takes time linear in the size of the grammar, the image, and the complexity of one graph cut. Empirically, I showed that SFG-Parse achieves accuracies and energies comparable to α-expansion, which returns optima within a constant factor of the global optimum, while taking exponentially less time to do so, and that SFGs are able to exploit the natural compositional structure of images to improve scene understanding. The work in this chapter also appears in Friesen and Domingos [46].

An important direction for future work is to develop and implement parameter and structure learning algorithms for SFGs, as their unique combination of tractability and expressivity provides a powerful method for better understanding natural scenes. This may require creating a dataset of parsed images that can be used for training SFGs. Beyond scene understanding, the unique combination of decomposability and submodularity exploited by SFGs should also benefit many other domains, including activity recognition, social network modeling, and probabilistic knowledge bases.

Decomposability can also be used to directly develop novel learning algorithms. In

the next chapter, I develop a novel algorithm for learning deep neural networks that uses combinatorial search to exploit decomposability in the learning objective in order to compute weight updates for models that cannot be learned with standard backpropagation-based methods.

Chapter 6

DEEP LEARNING AS A MIXED CONVEX-COMBINATORIAL OPTIMIZATION PROBLEM

6.1 Introduction

The original approach to neural classification was to learn single-layer models with hard-threshold activations, like the perceptron [121]. However, it proved difficult to extend these methods to multiple layers, because hard-threshold units, having derivative zero almost everywhere and being discontinuous at the origin, cannot be used with backpropagation [123]. Instead, the community turned to multilayer networks with soft activation functions, such as the sigmoid and, more recently, the ReLU, for which gradients can be computed efficiently by backpropagation.

This approach has enjoyed remarkable success, enabling researchers to train networks with hundreds of layers and learn models that have significantly higher accuracy on a variety of tasks than any previous approach. However, as networks become deeper and wider, there has been a growing trend towards using hard-threshold activations for quantization purposes, where they enable binary or low-precision inference (e.g., Hubara et al. [64], Lin and Talathi [95], Rastegari et al. [116], Zhou et al. [155], Zhu et al. [156]) and training (e.g., Li et al. [94], Lin et al. [96], Micikevicius et al. [101], Tang et al. [138]), which can greatly reduce the energy, memory, and computation required by modern deep networks. Beyond quantization, the scale of the output of hard-threshold units is independent of (or insensitive to) the scale of their input, which can alleviate vanishing and exploding gradient issues and should help avoid some of the pathologies that occur during low-precision training with backpropagation [94]. Avoiding these issues is crucial for developing large systems of deep networks that can be used to perform even more

complex tasks.

For these reasons, it is important to develop well-motivated and efficient techniques for learning deep neural networks with hard-threshold units. In this chapter, I propose a framework for learning deep hard-threshold networks that stems from the observation that hard-threshold units output discrete values, indicating that combinatorial optimization may provide a principled method for training these networks. By specifying a set of discrete targets for each hidden-layer activation, the network decomposes into many individual perceptrons, each of which can be trained easily given its inputs and targets. The difficulty in learning a deep hard-threshold network is thus in setting the targets so that each trained perceptron – including the output units – has a linearly separable problem to solve and thus can achieve its targets. I show that networks in which this is possible can be learned using a novel mixed convex-combinatorial optimization framework.

Unlike in Chapter 4, where setting the values of a subset of a function's variables decomposes that function, the functions being optimized in deep learning do not decompose for any setting of their variables (i.e., the network weights) and thus RDIS is not directly applicable. However, the learning problem does decompose as a result of introducing and setting the aforementioned target variables. While this may also be useful for computational reasons (e.g., see Carreira-Perpiñán and Wang [22] and Taylor et al. [140]), its main use in this chapter is to define a framework that enables computing weight updates for learning without needing to backpropagate gradients through the hard-threshold units.

Building on this framework, I then develop a recursive algorithm, feasible target propagation (FTPROP), for learning deep hard-threshold networks. Since this is a discrete optimization problem, I develop heuristics for setting the targets based on per-layer loss functions. The mini-batch version of FTPROP can be used to explain and justify the oft-used straight-through estimator [14, 60], which can now be seen as an instance of FTPROP with a specific choice of per-layer loss function and target heuristic. Finally, I develop a novel per-layer loss function that improves learning of deep hard-threshold networks. Empirically, I show improvements for my algorithm over the straight-through estimator on CIFAR-10

for two convolutional networks and on ImageNet for AlexNet and ResNet-18, with multiple types of hard-threshold activation.

Related Work

The most common method for learning deep hard-threshold networks is to use back-propagation with the straight-through estimator (STE) [14, 60], which simply replaces the derivative of each hard-threshold unit with the identity function. The STE is used in the quantized network literature (see citations above) to propagate gradients through quantized activations, and is used in Shalev-Shwartz et al. [130] for training with flat activations. Later work generalized the STE to replace the hard-threshold derivative with other functions, including saturated versions of the identity function [64]. However, while the STE tends to work quite well in practice, I know of no rigorous justification or analysis of why it works or how to choose replacement derivatives. Beyond being unsatisfying in this regard, the STE is not well understood and can lead to gradient mismatch errors, which compound as the number of layers increases [95]. I show here that the (saturated) STE is a special case of my framework, thus providing a principled justification for it and a basis for exploring and understanding alternatives.

Another common approach to training with hard-threshold units is to use randomness, either via stochastic neurons (e.g., Bengio et al. [14], Hubara et al. [64]) or probabilistic training methods, such as those of Soudry et al. [137] or Williams [148], both of which are methods for softening hard-threshold units. In contrast, my goal is to learn networks with deterministic hard-threshold units.

Finally, target propagation [13, 22, 88, 89, 91, 140] is a method that explicitly associates a target with the output of each activation in the network, and then updates each layer's weights to make its activations more similar to the targets. My framework can be viewed as an instance of target propagation that uses combinatorial optimization to set discrete targets, whereas previous approaches employed continuous optimization. The MADALINE Rule II (MRII) algorithm [150] can also be seen as a special case of my framework and of target

propagation, where only one target is set at a time.

6.2 Learning Deep Networks with Hard-Threshold Units

Given a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{m}$ with vector-valued inputs $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and binary targets $t \in \{-1, +1\}$, the task is to learn an ℓ-layered deep neural network with hard-threshold units

$$y = f(\mathbf{x}; W) = g(W_\ell \, g(W_{\ell-1} \dots g(W_1 \mathbf{x}) \dots)), \tag{6.1}$$

with weight matrices $W = \{W_d : W_d \in \mathbb{R}^{n_d \times n_{d-1}}\}_{d=1}^{\ell}$ and element-wise activation function $g(\mathbf{x}) = \text{sign}(\mathbf{x})$, where sign is the sign function such that $\text{sign}(x) = 1$ if $x > 0$ and -1 otherwise. Each layer d has n_d units, where I define $n_0 = n$ for the input layer and let $\mathbf{h}_d = g(W_d \dots g(W_1 \mathbf{x}) \dots)$ denote the output of each hidden layer, where $\mathbf{h}_d = (h_{d1}, \dots, h_{dn_d})$ and $h_{dj} \in \{-1, +1\}$ for each layer d and each unit j. Similarly, I let $\mathbf{z}_d = W_d \, g(\dots g(W_1 \mathbf{x}) \dots)$ denote the pre-activation output of layer d. For compactness, I have incorporated the bias term into the weight matrices. I denote a row or column of a matrix W_d as $W_{d,:j}$ and $W_{d,j:}$, respectively, and the entry in the jth row and kth column as $W_{d,jk}$. Using matrix notation, this model can be written as $Y = f(X; W) = g(W_\ell \dots g(W_1 X) \dots)$, where X is the $n \times m$ matrix of dataset instances and Y is the $n_\ell \times m$ matrix of outputs. I let T_ℓ denote the matrix of final-layer targets, H_d denote the $n_d \times m$ matrix of hidden activations at layer d, and Z_d denote the $n_d \times m$ matrix of pre-activations at layer d. My goal will be to learn f by finding the weights W that minimize an aggregate loss $L(Y, T_\ell) = \sum_{i=1}^{m} L(y^{(i)}, t^{(i)})$ for some convex per-instance loss $L(y, t)$.

In the simplest case, a hard-threshold network with no hidden layers is a perceptron $Y = g(W_1 X)$, as introduced by Rosenblatt [121]. The goal of learning a perceptron, or any hard-threshold network, is to classify unseen data. A useful first step is to be able to correctly classify the training data, which I focus on here for simplicity when developing my framework; however, standard generalization techniques such as regularization are

easily incorporated into this framework. Since a perceptron is a linear classifier, it is only able to separate a linearly-separable dataset.

Definition 6.1. *A dataset* $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^{m}$ *is* linearly separable *iff there exists a vector* $\mathbf{w} \in \mathbb{R}^n$ *and a real number* $\gamma > 0$ *such that* $(\mathbf{w} \cdot \mathbf{x}^{(i)})t^{(i)} > \gamma$ *for all* $i = 1 \dots m$.

When a dataset is linearly separable, the perceptron algorithm is guaranteed to find its separating hyperplane in a finite number of steps [110], where the number of steps required is dependent on the size of the margin γ. However, linear separability is a very strong condition, and even simple functions, such as XOR, are not linearly separable and thus cannot be learned by a perceptron [103]. It is thus important to be able to learn multilayer hard-threshold networks.

Consider a simple single-hidden-layer hard-threshold network

$$Y = f(X; W) = g(W_2\, g(W_1 X)) = g(W_1 H_1)$$

for a dataset $\mathcal{D} = (X, T_2)$, where $H_1 = g(W_1 X)$ are the hidden-layer activations. Clearly, Y and H_1 are both collections of (single-layer) perceptrons. Backpropagation cannot be used to train the input layer's weights W_1, but since each hidden activation h_{1j} is the output of a perceptron, then if the value $t_{1j} \in \{-1, +1\}$ that each hidden unit *should* take for each input $\mathbf{x}^{(i)}$ was known, a learning algorithm could use the perceptron algorithm to set W_1 to produce these values. I refer to t_{1j} as the *target* of h_{1j}. Given a matrix of hidden-layer targets $T_1 \in \{-1, +1\}^{n_1 \times m}$, each layer (and in fact each perceptron in each layer) can be learned separately, as they no longer depend on each other, where the goal of perceptron learning is to update the weights of each layer d so that its activations H_d equal its targets T_d given inputs T_{d-1}. Figure 6.1 shows an example of this decomposition. I denote the targets of an ℓ-layer network as $T = \{T_1, \dots, T_\ell\}$, where T_k for $k = 1 \dots \ell - 1$ are the hidden-layer targets, T_ℓ are the dataset targets, and I often let $T_0 = X$ for notational convenience.

Auxiliary-variable-based approaches, such as ADMM [22, 140] and other target propagation methods [88, 91] use a similar process for decomposing the layers of a network;

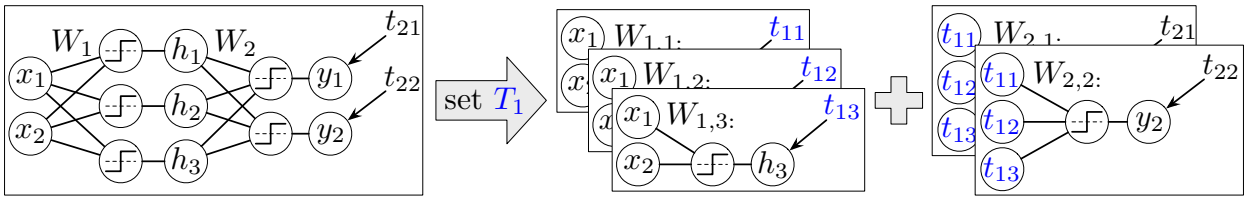


Figure 6.1: After setting the hidden-layer targets T_1 of a deep hard-threshold network, the network decomposes into independent perceptrons, which can then be learned with standard methods.

however, these focus on continuous variables and impose (soft) constraints to ensure that each activation equals its auxiliary variable. I take a different approach here, inspired by the combinatorial nature of the problem and the perceptron algorithm.

Since the final layer is a perceptron, the training instances can only be separated if the hidden-layer activations H_1 are linearly separable with respect to the dataset targets T_2. Thus, the hidden-layer targets T_1 must be set such that they are linearly separable with respect to the dataset targets T_2, since the hidden-layer targets T_1 are the intended values of their activations H_1. However, in order to ensure that the hidden-layer activations H_1 will equal their targets T_1 after training, the hidden-layer targets T_1 must be able to be produced (exactly) by the first layer, which is only possible if the hidden-layer targets T_1 are linearly separable with respect to the inputs X. Thus, a sufficient condition for $f(X; W)$ to separate the data is that the hidden-layer targets induce linear separability in all units in both layers of the network. I refer to this property as *feasibility.*

Definition 6.2. *A setting of the targets $T = \{T_1, \ldots, T_\ell\}$ of an ℓ-layer deep hard-threshold network $f(X; W)$ is* feasible *for a dataset $\mathcal{D} = (X, T_\ell)$ iff for each unit $j = 1 \ldots n_d$ in each layer $d = 1 \ldots \ell$ the dataset formed by its inputs T_{d-1} and targets $T_{d,j:}$ is linearly separable, where $T_0 \triangleq X$.*

Feasibility is a much weaker condition than linear separability, since the output decision boundary of a multilayer hard-threshold network with feasible targets is in general highly nonlinear. It follows from the definition of feasibility and convergence of the perceptron algorithm that if a feasible setting of a network's targets on a dataset exists, the network

can separate the training data.

Proposition 6.1. *Let $\mathcal{D} = \{(\mathbf{x}^{(i)}, t^{(i)})\}$ be a dataset and let $f(X; W)$ be an ℓ-layer hard-threshold network with feasible targets $T = \{T_1, \ldots, T_\ell\}$ in which each layer d of f was trained separately with inputs T_{d-1} and targets T_d, where $T_0 \triangleq X$, then f will correctly classify each instance $\mathbf{x}^{(i)}$, such that $f(\mathbf{x}^{(i)}; W)t^{(i)} > 0$ for all $i = 1 \ldots m$.*

Learning a deep hard-threshold network thus reduces to finding a feasible setting of its targets and then optimizing its weights given these targets, i.e., mixed convex-combinatorial optimization. The simplest method for this is to perform exhaustive search on the targets. Exhaustive search iterates through all possible settings of the hidden-layer targets, updating the weights of each perceptron whose inputs or targets changed, and returns the weights and feasible targets that result in the lowest loss. While impractical, exhaustive search is worth briefly examining to better understand the solution space. In particular, because of the decomposition afforded by setting the targets, exhaustive search over just the targets is sufficient to learn the globally optimal deep hard-threshold network, even though the weights are learned by gradient descent.

Proposition 6.2. *If a feasible setting of a deep hard-threshold network's targets on a dataset $\mathcal{D}$ exists, then exhaustive search returns the global minimum of the loss in time exponential in the number of hidden-unit targets.*

Learning can be improved and feasibility relaxed if, instead of the perceptron algorithm, a more robust method is used for perceptron learning. For example, a perceptron can be learned for a non-linearly-separable dataset by minimizing the hinge loss $L(z, t) = \max(0, 1 - tz)$, a convex loss on the perceptron's pre-activation output z and target t that maximizes the margin when combined with L2 regularization. In general, however, any method for learning linear classifiers can be used. I denote the loss used to train the weights of a layer d as L_d, where the loss of the final layer L_ℓ is the output loss.

At the other end of the search spectrum is hill climbing. In each iteration, hill climbing evaluates all neighboring states of the current state (i.e., target settings that differ from

the current one by only one target) and chooses the one with the lowest loss. The search halts when none of the new states improve the loss. Each state is evaluated by optimizing the weights of each perceptron given the state's targets, and then computing the output loss. Hill climbing is more practical than exhaustive search, since it need not explore an exponential number of states, and it also provides the same local optima guarantee as gradient descent on soft-threshold networks.

Proposition 6.3. *Hill climbing on the targets of a deep hard-threshold network returns a local minimum of the loss, where each iteration takes time linear in the size of the set of proposed targets.*

Exhaustive search and hill climbing comprise two ends of the discrete optimization spectrum. Beam search, which maintains a beam of the most promising solutions and explores each, is another powerful approach that contains both hill climbing and exhaustive search as special cases. In general, however, any discrete optimization algorithm can be used for setting targets. For example, methods from satisfiability solving, integer linear programming, or constraint satisfaction might work well, as the linear separability requirements of feasibility can be viewed as constraints on the search space.

I believe that my mixed convex-combinatorial optimization framework opens many new avenues for developing learning algorithms for deep networks, including those with non-differentiable modules. In the following section, I use these ideas to develop a learning algorithm that hews much closer to standard methods, and in fact contains the straight-through estimator as a special case.

6.3 Feasible Target Propagation

The open question from the preceding section is how to set the hidden-layer targets. Generating good, feasible targets for the entire network at once is a difficult problem; instead, an easier approach is to propose targets for only one layer at a time. As in backpropagation, it is natural to begin at the output layer, since the final-layer targets are given, and successively set targets for each upstream layer. Further, since it is hard to

know a priori if a setting of a layer's targets is feasible for a given network architecture, a simple alternative is to set the targets for a layer d and then optimize the upstream weights (i.e., weights in layers $j \leq d$) to check if the targets are feasible. Since the goal when optimizing a layer's weights and when setting its upstream targets (i.e., its inputs) is the same – namely, to induce feasibility – a natural method for setting target values is to choose targets that reduce the layer's loss L_d. However, because the targets are discrete, moves in target space are large and non-smooth and cannot be guaranteed to lower the loss without actually performing the move. Thus, heuristics are necessary. I discuss these in more detail below. Determining feasibility of the targets at layer d can be done by recursively updating the weights of layer d and proposing targets for layer $d-1$ given the targets for layer d. This recursion continues until the input layer is reached, where feasibility (i.e., linear separability) can be easily determined by optimizing that layer's weights given its targets and the dataset inputs. The targets at layer d can then be updated based on the information gained from the recursion and, if the upstream weights were altered, based on the new outputs of layer $d-1$. I call this recursive algorithm *feasible target propagation*, or FTPROP. Pseudocode is shown in Algorithm 6.

As the name implies, FTPROP is a form of target propagation [88, 89, 91] that uses discrete instead of continuous optimization to set targets. FTPROP is also highly related to RDIS (Chapter 4), While RDIS is applied only to continuous problems, the ideas behind RDIS can be generalized to discrete variables via the sum-product theorem (Chapter 3). Since RDIS is based on satisfiability (SAT) solvers, this suggests an interesting connection between FTPROP and SAT that I leave for future work.

Of course, modern deep networks will not always have a feasible setting of their targets for a given dataset. For example, a convolutional layer imposes a large amount of structure on its weight matrix, making it less likely that the layer's input will be linearly separable with respect to its targets. Further, ensuring feasibility will in general cause learning to overfit the training data, which will worsen generalization performance. Thus, it is important to be able to relax the feasibility requirements of FTPROP.

Algorithm 6 Train an ℓ-layer hard-threshold network $Y = f(X; W)$ on dataset $\mathcal{D} = (X, T_\ell)$ with feasible target propagation (FTPROP) using loss functions $L = \{L_d\}_{d=1}^{\ell}$.

initialize weights $W = \{W_1, \ldots, W_\ell\}$ randomly
initialize targets $T_1, \ldots, T_{\ell-1}$ as the outputs of their hidden units in $f(X; W)$
set $T_0 \leftarrow X$ and set $T \leftarrow \{T_0, T_1, \ldots, T_\ell\}$
FTPROP(W, T, L, ℓ) *// train the network by searching for a feasible target setting*

function FTPROP(weights W, targets T, losses L, and layer index d)
    optimize W_d with respect to layer loss $L_d(Z_d, T_d)$ *// check feasibility; $Z_d = W_d T_{d-1}$*
    if activations $H_d = g(W_d T_{d-1})$ equal the targets T_d **then return** True *// feasible*
    else if this is the first layer (i.e., $d = 1$) **then return** False *// infeasible*
    while computational budget of this layer not exceeded **do** *// e.g., set by beam search*
        $T_{d-1} \leftarrow$ heuristically set targets for upstream layer to reduce layer loss $L_d(Z_d, T_d)$
        if FTPROP$(W, T, L, d-1)$ **then** *// check if targets T_{d-1} are feasible*
            optimize W_d with respect to layer loss $L_d(Z_d, T_d)$
            if activations $H_d = g(W_d T_{d-1})$ equal the targets T_d **then return** True *// feasible*
    return False

In addition, there are many benefits of using mini-batch instead of full-batch training, including improved generalization gap (e.g., see LeCun et al. [90] or Keskar et al. [71]), reduced memory usage, the ability to exploit data augmentation, and the prevalence of libraries and hardware designed for it.

Fortunately, it is straightforward to convert FTPROP to a mini-batch algorithm and to relax the feasibility requirements. In particular, since it is important not to overcommit to any one mini-batch, the mini-batch version of FTPROP (i) only updates the weights and targets of each layer once per mini-batch; (ii) only takes a small gradient step on each layer's weights, instead of optimizing them fully; (iii) sets the targets of the downstream layer in parallel with updating the current layer's weights, since the weights will not change much; and (iv) removes all checks for feasibility. I call this algorithm FTPROP-MB and present pseudocode in Algorithm 7. FTPROP-MB closely resembles backpropagation-based methods, allowing us to easily implement it with standard GPU-based libraries.

Algorithm 7 Train an ℓ-layer hard-threshold network $Y = f(X; W)$ on dataset $\mathcal{D} = (X, T_\ell)$ with mini-batch feasible target propagation (FTPROP-MB) using loss functions $L = \{L_d\}_{d=1}^{\ell}$.

initialize weights $W = \{W_1, \ldots, W_\ell\}$ randomly
for each minibatch (X_b, T_b) from $\mathcal{D}$ **do**
    initialize targets $T_1, \ldots, T_{\ell-1}$ as the outputs of their hidden units in $f(X_b; W)$
    set $T_0 \leftarrow X_b$, set $T_\ell \leftarrow T_b$, and set $T \leftarrow \{T_0, \ldots, T_\ell\}$
    FTPROP-MB(W, T, L, ℓ)

function FTPROP-MB(weights W, targets T, losses L, and layer index d)
    $\hat{T}_{d-1} \leftarrow$ heuristically set targets for upstream layer to reduce layer loss $L_d(Z_d, T_d)$
    update W_d with respect to layer loss $L_d(Z_d, T_d)$ // *where* $Z_d = W_d T_{d-1} = W_d H_{d-1}$
    if $d > 1$ **then** FTPROP-MB$(W,\ \{T_0, \ldots, \hat{T}_{d-1}, \ldots, T_\ell\},\ L,\ d-1)$

6.3.1 Target Heuristics

When the activations of each layer are differentiable, backpropagation provides a method for telling each layer how to adjust its outputs to improve the loss. Conversely, in hard-threshold networks, target propagation provides a method for telling each layer how to adjust its outputs to improve the next layer's loss, as long as the targets are set effectively. While gradients cannot propagate through hard-threshold units, the derivatives within a layer can still be computed. An effective and efficient heuristic for setting the target t_{dj} for an activation h_{dj} of layer d is to use the (negative) sign of the partial derivative of the next layer's loss. Specifically, I set $t_{dj} = r(h_{dj})$, where

$$r(h_{dj}) \triangleq \operatorname{sign}\left(-\frac{\partial}{\partial h_{dj}} L_{d+1}(Z_{d+1}, T_{d+1})\right) \tag{6.2}$$

and Z_{d+1} is either the pre-activation or post-activation output, depending on the choice of loss.

When used to update only a single target at a time, this heuristic will often set the target value that correctly results in the lowest loss. In particular, when L_{d+1} is convex, its negative partial derivative with respect to h_{dj} by definition points in the direction of the

global minimum of L_{d+1}. Without loss of generality, let $h_{dj} = -1$. Now, if $r(h_{dj}) = -1$, then it follows from the convexity of the loss that flipping h_{dj} and keeping all other variables the same would increase L_{d+1}. On the other hand, if $r(h_{dj}) = +1$, then flipping h_{dj} may or may not reduce the loss, since convexity cannot tell us which of $h_{dj} = +1$ or $h_{dj} = -1$ results in a smaller L_{d+1}. However, the discrepancy between h_{dj} and $r(h_{dj})$ indicates a lack of confidence in the current value of h_{dj}. A natural choice is thus to set t_{dj} to push the pre-activation value of h_{dj} towards 0, making h_{dj} more likely to flip. Setting $t_{dj} = r(h_{dj}) = +1$ accomplishes this. I note that, while this heuristic performs well, there is still room for improvement, for example by extending $r(\cdot)$ to better handle the $h_{dh} \neq r(h_{dj})$ case or by combining information across the batch. I leave such investigations for future work.

6.3.2 Layer Loss Functions

The hinge loss, shown in Figure 6.2a, is a robust version of the perceptron criterion and is thus a natural per-layer loss function to use for finding a feasible (or nearly feasible) setting of the targets and weights. However, in preliminary experiments, I found that learning tended to stall and become erratic over time when using the hinge loss for each layer. I attribute this to two separate issues. First, the hinge loss is sensitive to noisy data and outliers [151], which can cause learning to focus on instances that are unlikely to ever be classified correctly, instead of on instances near the separator. Second, since with convolutional layers and large, noisy datasets it is unlikely that a layer's inputs are entirely linearly separable, it is thus important to prioritize some targets over others. Ideally, the highest priority targets would be those with the largest effect on the output loss.

The first issue can be solved by saturating (truncating) the hinge loss, thus making it less sensitive to outliers [151]. The saturated hinge loss, shown in Figure 6.2b, is $\text{sat\_hinge}(z, t; b) = \max(0, 1 - \max(tz, b))$ for some threshold b, where I set $b = -1$ to make its derivative symmetric. The second problem can be solved in a variety of ways, including randomly subsampling targets or weighting the loss associated with each target according to some heuristic. The simplest and most accurate method that I have found is to weight

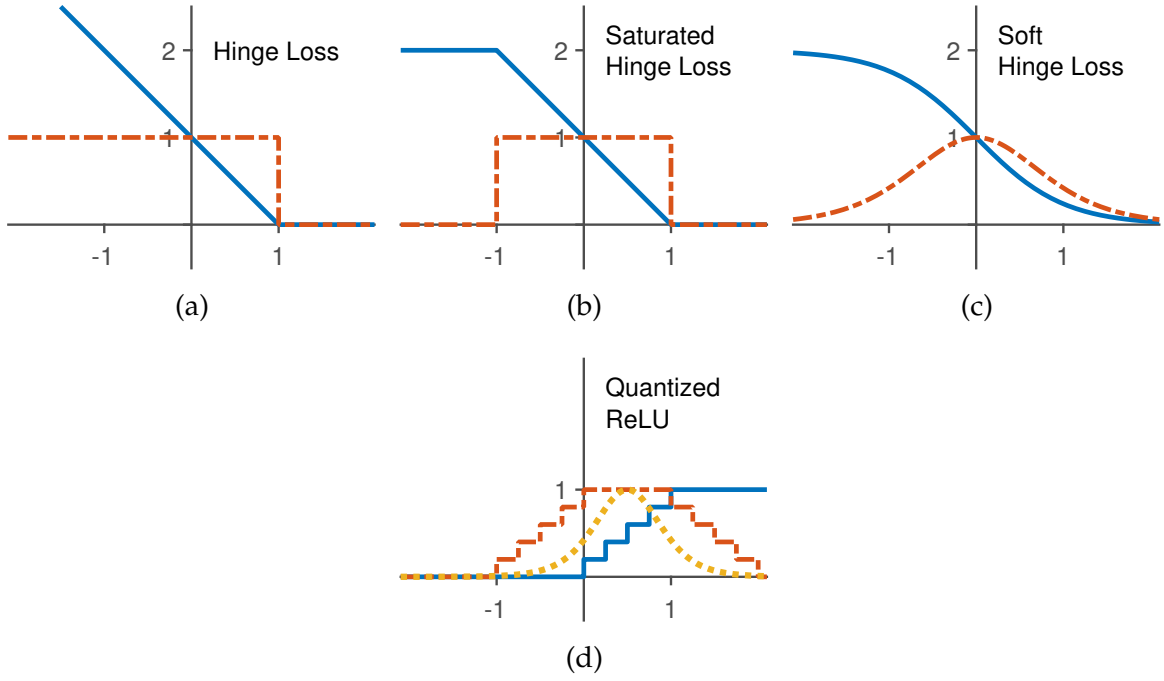


Figure 6.2: Figures (a)-(c) show different per-layer loss functions (solid blue line) and their derivatives (dashed red line). Figure (d) shows the quantized ReLU activation (solid blue line), which is a sum of step functions, its corresponding sum of saturated-hinge-loss derivatives (dashed red line), and the soft-hinge-loss approximation to this sum that was found to work best (dotted yellow line).

the loss for each target t_{dj} by the magnitude of the partial derivative of the next layer's loss L_{d+1} with respect to the target's hidden unit h_{dj}, such that

$$L_d(z_{dj}, t_{dj}) = \text{sat\_hinge}(z_{dj}, t_{dj}) \cdot \left| \frac{\partial L_{d+1}}{\partial h_{dj}} \right|. \tag{6.3}$$

While the saturated hinge loss works well, if the input z_{dj} ever moves out of the range $[-1, +1]$ then its derivative will become zero and the unit will no longer be trainable. To avoid this, I propose the *soft hinge loss*, shown in Figure 6.2c, where $\text{soft\_hinge}(z, t) = \tanh(-zt)+1$. Like the saturated hinge, the soft hinge has slope 1 at the threshold and has a symmetric derivative; however, it also benefits from having a larger input region with non-zero

derivative. Note that Bengio et al. [14] report that using the derivative of a sigmoid as the STE performed worse than the identity function. Based on my experiments with other loss functions, including variations of the squared hinge loss and the log loss, this is most likely because the slope of the sigmoid is less than unity at the threshold, which causes vanishing gradients. Loss functions with asymmetric derivatives around the threshold also seemed to perform worse than those with symmetric derivatives (e.g., the saturating and soft hinge losses). In my experiments, I show that the soft hinge loss outperforms the saturated hinge loss for both sign and quantized-ReLU activations, which I discuss below.

6.3.3 Relationship to the Straight-Through Estimator

When each loss term in each hidden layer is scaled by the magnitude of the partial derivative of its downstream layer's loss and each target is set based on the sign of the same partial derivative, then target propagation transmits information about the output loss to every layer in the network, despite the hard-threshold units. Interestingly, this combination of loss function and target heuristic can exactly reproduce the weight updates of the straight-through estimator (STE). Specifically, the weight updates that result from using the scaled saturated hinge loss from (6.3) and the target heuristic in (6.2) are exactly those of the saturated straight-through estimator (SSTE) defined in Hubara et al. [64], which replaces the derivative of $\mathrm{sign}(z)$ with $1_{|z|\leq 1}$, where $1_{(\cdot)}$ is the indicator function. Other STEs correspond to different choices of per-layer loss function. This connection provides a justification for existing STE approaches, which can now be seen as instances of FTPROP with a specific choice of per-layer loss function and target heuristic. I believe that this will enable more-principled investigations and extensions of these methods in future work.

6.3.4 Quantized Activations

Straight-through estimation is also commonly used to backpropagate through quantized variants of standard activations, such as the ReLU. Figure 6.2d shows a quantized ReLU

(qReLU) with 6 evenly spaced thresholds. The simplest and most popular straight-through estimator (STE) for qReLU is to use the derivative of the saturated (or clipped) ReLU $\frac{\partial \,\mathrm{sat\_ReLU}(x)}{\partial x} = 1_{0<x<1}$, where $\mathrm{sat\_ReLU}(x) = \min(1, \max(x, 0))$. However, instead consider the qReLU activation from the viewpoint of FTPROP. From this perspective, the qReLU is actually a (normalized) sum of step functions

$$\mathrm{qReLU}(z) = \frac{1}{k} \sum_{i=0}^{k-1} \mathrm{step}(z - \frac{i}{k-1}),$$

where $\mathrm{step}(z) = 1$ if $z > 0$ and 0 otherwise, and is a linear transformation of $\mathrm{sign}(z)$. The resulting derivative of the sum of saturated hinge losses (one for each step function) is shown in red in Figure 6.2d, and is clearly quite different than the STE described above. In initial experiments, this performed as well as or better than the STE; however, I achieved additional performance improvements by using the softened approximation shown in yellow in Figure 6.2d, which is simply the derivative of a soft hinge that has been scaled and shifted to match the qReLU domain. This is a natural choice because the derivative of a sum of a small number of soft hinge losses has a shape similar to that of the derivative of a single soft hinge loss.

6.4 Empirical Evaluation of Feasible Target Propagation

I evaluated FTPROP-MB with soft hinge per-layer losses (FTP-SH) for training deep networks with sign and 2- and 3-bit qReLU activations by comparing models trained with FTP-SH to those trained with the saturated straight-through estimators (SSTEs) described earlier (although, as discussed, these SSTEs can also be seen as instances of FTPROP-MB). I also trained each model with full-precision ReLU and saturated ReLU activations as baselines. I did not use weight quantization because my main interest is training with hard-threshold activations, and because recent work has shown that weights can be quantized with little effect on performance [64, 116, 155]. I tested these training methods on the CIFAR-10 [79] and ImageNet (ILSVRC 2012) [124] datasets. On CIFAR-10, I trained a simple 4-layer

	Sign		qReLU		ReLU	Saturated ReLU
	SSTE	*FTP-SH*	*SSTE*	*FTP-SH*		
4-layer convnet (CIFAR-10)	80.6	**81.3**	85.6	85.5	86.5	**87.3**
8-layer convnet (CIFAR-10)	84.6	**84.9**	88.4	**89.8**	91.2	91.2
AlexNet (ImageNet)	46.7	**47.3**	59.4	**60.7**	61.3	**61.9**
ResNet-18 (ImageNet)	**49.1**	47.8	60.6	**64.3**	**69.1**	66.9

Table 6.1: The best top-1 test accuracy for each network over all epochs when trained with sign, qReLU, and full-precision activations on CIFAR-10 or ImageNet. The hard-threshold activations are trained with FTPROP-MB with per-layer soft hinge losses (FTP-SH) and the saturated straight-through estimator (SSTE). Bold numbers denote the best performing activation in each pair.

convolutional network and the 8-layer convolutional network of Zhou et al. [155]. On ImageNet, I trained AlexNet [80], the most common model in the quantization literature, and ResNet-18 [57]. Experiment details are provided below in Section 6.4.3, along with learning curves for all experiments.

6.4.1 CIFAR-10 Results

Test accuracies for the 4-layer and 8-layer convolutional network on CIFAR-10 are shown in Table 6.1. For the simpler 4-layer model, FTP-SH shows a consistent 0.5-1% accuracy gain over SSTE for the entire training trajectory, resulting in the 0.7% improvement shown in Table 6.1. However, for the 2-bit qRELU activation, SSTE and FTP-SH perform nearly identically in the 4-layer model. Conversely, for the more complex 8-layer model, the FTP-SH accuracy is only 0.3% above SSTE, but for the qReLU activation FTP-SH achieves a consistent 1.4% improvement over SSTE.

I posit that the decrease in performance gap for the sign activation when moving from the 4- to 8-layer model is because both methods are able to effectively train the higher-capacity model to achieve close to its best possible performance on this dataset, whereas the opposite is true for the qReLU activation; i.e., the restricted capacity of the 4-layer

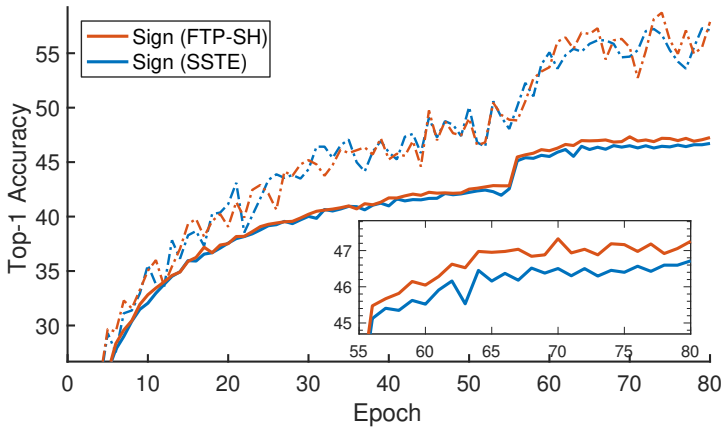
model limits the ability of both methods to train the more expressive qReLU effectively. If this is true, then I expect that FTP-SH will outperform SSTE for both the sign and qReLU activations on a harder dataset. Unsurprisingly, none of the low-precision methods perform as well as the high-precision methods; however, the narrowness of the performance gap between 2-bit qReLU with FTP-SH and full-precision ReLU is encouraging.

6.4.2 ImageNet Results

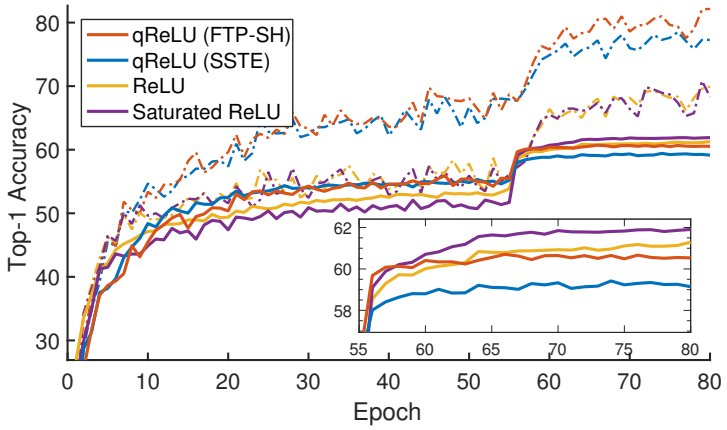
The results from the ImageNet experiments are also shown in Table 6.1. As predicted from the CIFAR-10 experiments, FTP-SH improves test accuracy on AlexNet for both sign and 2-bit qReLU activations on the more challenging ImageNet dataset. This is also shown in Figure 6.3, which plots the top-1 train and test accuracy curves for the six different activation functions for AlexNet on ImageNet. The left-hand plot shows that training sign activations with FTP-SH provides consistently better test accuracy than SSTE throughout the training trajectory, despite the hyperparameters being optimized for SSTE. This improvement is even larger for the 2-bit qReLU activation in the right-hand plot, where the FTP-SH qReLU even outperforms the full-precision ReLU for part of its trajectory, and outperforms the SSTE-trained qReLU by almost 2%. Interestingly, the saturated ReLU outperforms the standard ReLU by almost a full point of accuracy. I believe that this is due to the regularization effect caused by saturating the activation. This may also account for the surprisingly good performance of the FTP-SH qReLU relative to full-precision ReLU, as hard-threshold activations also provide a strong regularization effect.

Finally, I ran a single experiment with ResNet-18 on ImageNet, using hyperparameters set from previous works that used SSTE, to check (i) whether the soft hinge loss exhibits vanishing gradient behavior due to its diminishing slope away from the origin, and (ii) to evaluate the performance of FTP-SH for a less-quantized ReLU (I used $k = 5$ steps, which is less than the full range of a 3-bit ReLU). While FTP-SH does slightly worse than SSTE for the sign function, I believe that this is because the hyperparameters were tuned for SSTE and not due to vanishing gradients, as I would expect much worse accuracy in that case.

Results from the qReLU activation provide further evidence against vanishing gradients as FTP-SH for qReLU outperforms SSTE by almost 4% in top-1 accuracy (Table 6.1).



(a)



(b)

Figure 6.3: The top-1 train (thin dashed lines) and test (thick solid lines) accuracies for AlexNet with different activation functions on ImageNet. The inset figures show the test accuracy for the final 25 epochs in detail. In both figures, FTPROP-MB with soft hinge (FTP-SH, red) outperforms the saturated straight-through estimator (SSTE, blue). (a) shows the network with sign activations. (b) shows the network with variants of the ReLU activation. Interestingly, the 2-bit quantized ReLU (qReLU) trained with my method (FTP-SH) performs nearly as well as the full-precision ReLU. Further, saturated ReLU outperforms standard ReLU. Best viewed in color.

6.4.3 *Experiment Details*

All experiments were performed using PyTorch (`http://pytorch.org/`).

CIFAR-10 Experiments

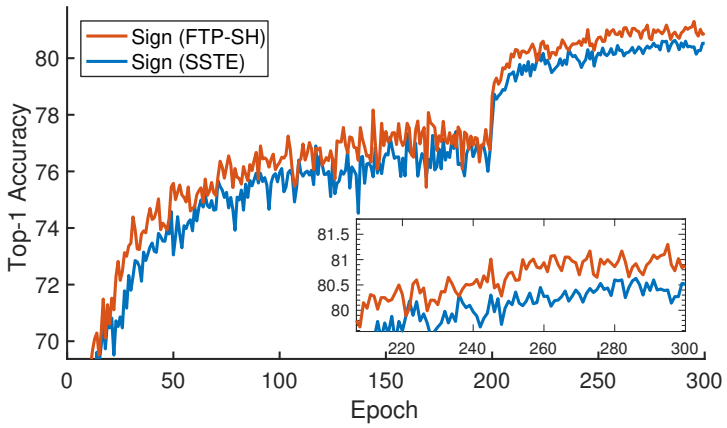
On CIFAR-10, which has 50K training images and 10K test images divided into 10 classes, I trained both a simple 4-layer convolutional network and a deeper 8-layer convolutional network used in [155] with the above methods and then compared their top-1 accuracies on the test set. I pre-processed the images with mean / std normalization, and augmented the dataset with random horizontal flips and random crops from images padded with 4 pixels. Hyperparameters were chosen based on a small amount of exploration on a validation set.

The first network I tested on CIFAR-10 was a simple 4-layer convolutional network (convnet) structured as: conv$(32) \to$ conv$(64) \to$ fc$(1024) \to$ fc(10), where conv(c) and fc(c) indicate a convolutional layer and fully-connected layer, respectively, with c channels. Both convolutional layers used 5×5 kernels. Max-pooling with stride 2 was used after each convolutional layer, and a non-linearity was placed before each of the above layers except the first. Adam [73] with learning rate 2.5e-4 and weight decay 5e-4 was used to minimize the cross-entropy loss for 300 epochs. The learning rate was decayed by a factor of 0.1 after 200 and 250 epochs.

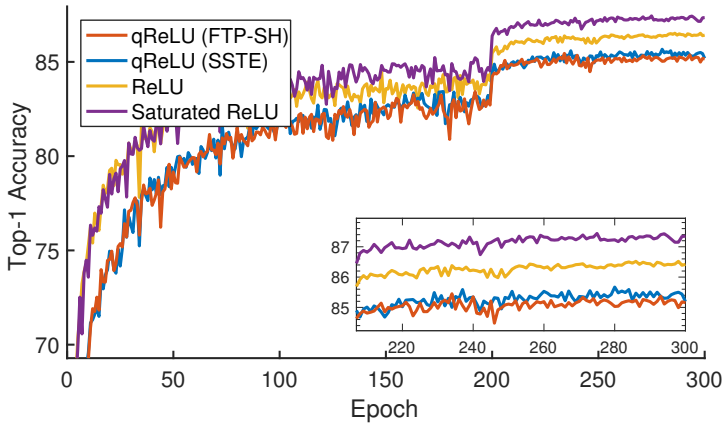
In order to evaluate the performance of FTPROP-MB with the soft hinge loss on a deeper network, I adapted the 8-layer convnet from Zhou et al. [155] to CIFAR-10. This network has 7 convolutional layers and one fully-connected layer for the output and uses batch normalization [65] before each non-linearity. I optimized the cross-entropy loss with Adam using a learning rate of 1e-3 and a weight decay of 1e-7 for the sign activation and 5e-4 for the qReLU and baseline activations. I trained for 300 epochs, decaying the learning rate by 0.1 after 200 and 250 epochs.

Learning Curves for CIFAR-10

Figures 6.4 and 6.5 show the learning curves for the 4-layer and 8-layer convolutional networks, respectively, when trained on CIFAR-10.

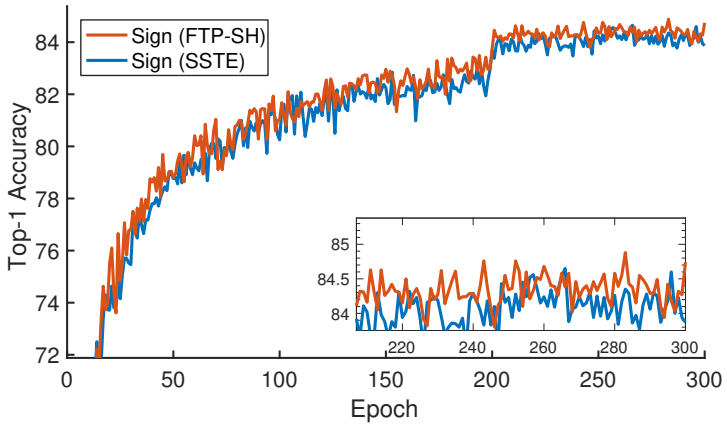


(a)

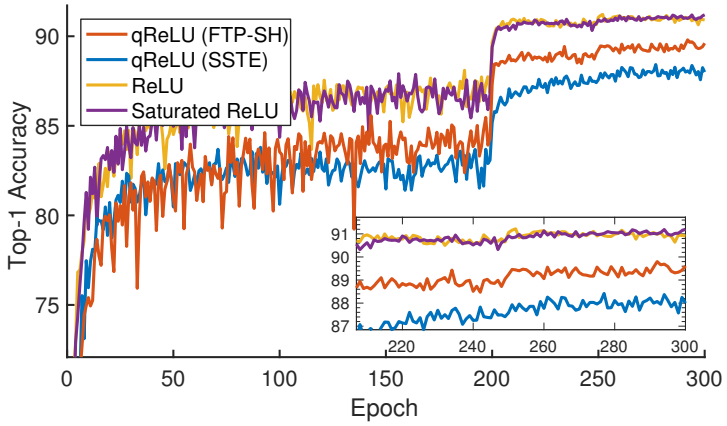


(b)

Figure 6.4: The top-1 test accuracies for the 4-layer convolutional network with different activation functions on CIFAR-10. The inset figures show the test accuracy for the final 100 epochs in detail. Figure (a) shows the network with sign activations. Figure (b) shows the network with 2-bit quantized ReLU (qReLU) activations and with the full-precision baselines. Best viewed in color.



(a)



(b)

Figure 6.5: The top-1 test accuracies for the 8-layer convolutional network with different activation functions on CIFAR-10. The inset figures show the test accuracy for the final 100 epochs in detail. (a) shows the network with sign activations. (b) shows the network with 2-bit quantized ReLU (qReLU) activations and with the full-precision baselines. Best viewed in color.

ImageNet (ILSVRC 2012) Experiments

On ImageNet, a much more challenging dataset with roughly 1.2M training images and 50K validation images divided into 1000 classes, I trained AlexNet, the most commonly

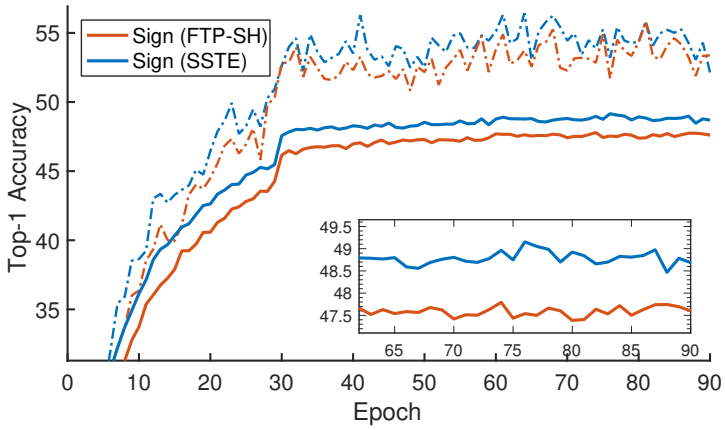
used model in the quantization literature, with different activations and compared top-1 and top-5 accuracies of the trained models on the validation set. As is standard practice, I treat the validation set as the test data. Images were resized to 256×256, mean / std normalized, and then randomly cropped to 224×224 and randomly horizontally flipped. Models are tested on centered 224×224 crops of the test images. Hyperparameters were set based on Zhou et al. [155] and Zhu et al. [156], which both used SSTE to train AlexNet on ImageNet.

I trained the Zhou et al. [155] variant of AlexNet [80] on ImageNet with sign, 2-bit qReLU, ReLU, and saturated ReLU activations. This version of AlexNet removes the dropout and replaces the local contrast normalization layers with batch normalization. My implementation does not split the convolutions into two separate blocks. I used the Adam optimizer with learning rate 1e-4 on the cross-entropy loss for 80 epochs, decaying the learning rate by 0.1 after 56 and 64 epochs. For the sign activation, I used a weight decay of 5e-6 as in Zhou et al. [155]. For the ReLU and saturated ReLU activations, which are much more likely to overfit, I used a weight decay of 5e-4, as used in Krizhevsky et al. [80]. For the qReLU activation, I used a weight decay of 5e-5, since it is more expressive than sign but less so than ReLU.

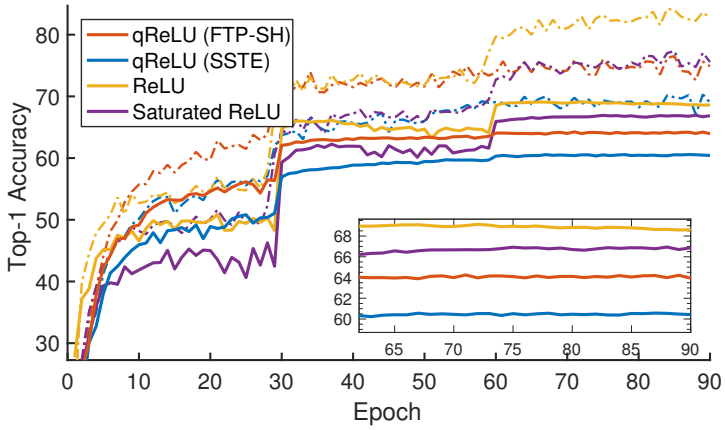
As with AlexNet, I trained ResNet-18 [58] on ImageNet with sign, qReLU, ReLU, and saturated ReLU activations; however, for ResNet-18, I used a qReLU with $k = 5$ steps (i.e., 6 quantization levels, requiring 3 bits). I used the ResNet code provided by PyTorch. I optimized the cross-entropy loss with SGD with learning rate 0.1 and momentum 0.9 for 90 epochs, decaying the learning rate by a factor of 0.1 after 30 and 60 epochs. For the sign activation, I used a weight decay of $5\text{e}{-7}$. For the ReLU and saturated ReLU activations, I used a weight decay of 1e-4. For the qReLU activation, I used a weight decay of 1e-5.

Learning Curves for ImageNet

Figures 6.3 (above) and 6.6 show the learning curves for the AlexNet and ResNet-18 networks, respectively, when trained on ImageNet.



(a)



(b)

Figure 6.6: The top-1 train (thin dashed lines) and test (thick solid lines) accuracies for ResNet-18 with different activation functions on ImageNet. The inset figures show the test accuracy for the final 60 epochs in detail. (a) shows the network with sign activations. (b) shows the network with 3-bit quantized ReLU (qReLU) activations and with the full-precision baselines. Best viewed in color.

6.5 Conclusion

In this chapter, I presented a novel mixed convex-combinatorial optimization framework for learning deep neural networks with hard-threshold units. Combinatorial optimization is used to set discrete targets for the hard-threshold hidden units, such that each unit only has a linearly-separable problem to solve. The network then decomposes into individual

perceptrons, which can be learned with standard convex approaches, given these targets. Based on this, I developed a recursive algorithm for learning deep hard-threshold networks, which I call feasible target propagation (FTPROP), and an efficient mini-batch version (FTPROP-MB). I showed that the commonly-used but poorly-justified saturating straight-through estimator (SSTE) is the special case of FTPROP-MB that results from using a saturated hinge loss at each layer and the target heuristic proposed above. Finally, I defined the soft hinge loss and showed that FTPROP-MB with a soft hinge loss at each layer improves classification accuracy for multiple models on CIFAR-10 and ImageNet when compared to the SSTE. This chapter demonstrates that decomposability is a general property that is useful beyond its effectiveness for defining new classes of tractable and expressive models.

In future work, I plan to develop novel target heuristics and layer loss functions by investigating connections between my framework, constraint satisfaction, and satisfiability. I also intend to further explore the benefits of deep networks with hard-threshold units. In particular, while recent research clearly shows their ability to reduce computation and energy requirements, they should also be less susceptible to vanishing and exploding gradients and may be less vulnerable to covariate shift and adversarial examples.

In the next chapter, I conclude and recap the contributions of this dissertation, and discuss extensions of the work presented in it.

Chapter 7
CONCLUSION

Decomposability is a simple property that may seem trivial or overly-restrictive at first glance, but is actually quite powerful and general. I have demonstrated in this dissertation that expressivity can be improved without sacrificing tractability by explicitly finding and exploiting fine-grained local decomposability structure as it varies throughout the space. Models that encode this approach within their representation and inference algorithms that adhere to it are often exponentially more efficient than standard methods, which they typically contain as special cases. In this chapter, I summarize my contributions and suggest promising directions for future work.

7.1 Contributions

In this work, I have sought to answer four questions regarding the relationship between tractability and expressivity, using a combination of analytical and empirical methods.

1. Is there a unifying condition or structure that ensures that inference, in its most general sense, is tractable? If so, how restrictive is this condition, with respect to both the expressivity and learnability of models that satisfy it?

In Chapter 3, I proved the sum-product theorem and a number of its corollaries, which, taken together, answer this question in the affirmative. Specifically, I showed that decomposability is a sufficient condition for tractable inference for any problem that consists of summing a function on a semiring, a class of problems that includes many of the most important and difficult tasks in artificial intelligence and machine learning, including probabilistic and logical inference, constraint satisfaction, nonconvex and combinatorial optimization, integration, and reasoning in (probabilistic) knowledge bases. Further, I proved that

models that enforce decomposability are strictly more expressive than models that require low treewidth to guarantee tractability, such as graphical models. Since exact inference is tractable in decomposable models, they are also easy to learn, and learned decomposable models can perform better than learned non-decomposable models. I demonstrated this empirically by learning a decomposable min-sum function for a challenging class of highly-multimodal functions and showing that optimizing this min-sum function produced much lower minima than optimizing the true (unlearned) test function, which does not account for inference complexity in its representation. Outside of my own work, the sum-product theorem has also been used by other researchers to develop tractable and expressive models, such as compositional kernel machines [49].

2. Even if decomposability is not overly restrictive theoretically, can models and algorithms that satisfy it be used to solve real problems?

To evaluate decomposability on real problems, in Chapter 4 I developed RDIS, a novel nonconvex optimization algorithm. RDIS recursively breaks the optimization problem into independent subproblems, thereby making it much easier to solve. I proved analytically that RDIS can find the global minimum exponentially faster than standard approaches because it exploits local decomposability. I then demonstrated this speed-up empirically on structure from motion and protein folding problems. I showed that RDIS is both faster and finds significantly better minima than standard methods for nonconvex optimization.

3. Can decomposability be used in conjunction with other types of structure to develop even more expressive models, or is it only useful in isolation?

In Chapter 5, I introduced submodular field grammars (SFGs) to demonstrate that decomposability and submodularity can be combined effectively. An SFG is a novel and expressive probabilistic model that uses a submodular Markov random field to encode the segmentation probability for every subregion and every production of every class in a stochastic context-free grammar. I used SFGs to define a novel type of image grammar in

which each instance of each class in the grammar can have arbitrary region shape. No previous image grammar formulation has been capable of representing or performing inference over all possible regions for each object in an image. To achieve this, I developed a novel move-making algorithm for approximate MAP inference in SFGs that takes exponentially less time than standard inference algorithms while returning comparable parses. As yet, learning SFGs has proved challenging, mainly due to a lack of training data and pre-existing image grammar structures on which to build. However, in preliminary experiments on scene understanding, SFGs resulted in higher-accuracy parses than a state-of-the-art deep network and that same network combined with a submodular MRF. This is because SFGs can better capture the compositional structure of images than competing models.

4. Finally, how generally applicable is decomposability and can it be used for tasks other than defining tractable representations?

In Chapter 6, I used decomposability to develop a novel algorithm, feasible target propagation, for learning deep neural networks with hard-threshold activations. Feasible target propagation uses combinatorial search to decompose the layers of the network in order to avoid backpropagating through the hard-threshold activations, which have derivative zero almost everywhere and thus cannot be used with backpropagation. I showed that the straight-through estimator, the standard method for training deep hard-threshold networks, is a special case of feasible target propagation. Further, I showed empirically that networks learned with feasible target propagation perform significantly better at image classification than those trained with the straight-through estimator for multiple network architectures on both the CIFAR-10 and ImageNet datasets and for both sign and quantized-ReLU activation functions.

7.2 Directions for Future Research: Unifying SPFs and Deep Learning

At the end of each of the preceding chapters, I described opportunities for extending the research presented in that chapter. In this section, I discuss promising directions for

extending the body of work contained in this dissertation as a whole.

Beyond the Semiring Abstraction

The sum-product theorem (Chapter 3) shows that the properties of a commutative semiring – associativity, commutativity, and distributivity – are sufficient to efficiently sum any decomposable function on a semiring over its domain. This is achieved in the proof of the sum-product theorem by using the semiring properties to "push" the summation over the function's domain (i.e., what I will call the marginalization operator here) from the root of the DAG representing the function to the leaves. The leaf functions are then summed over their domains (marginalized) and the function is evaluated using these instead.

The semiring abstraction has proven useful for characterizing SPFs and for unifying many core problems in AI and ML, but the concepts of tractable inference and marginalization are not tied to this abstraction. It is possible to extend the SPF framework to allow functions to consist of additional operators, and to allow the marginalization operator to be distinct from any of these operators. As long as it remains possible to push the marginalization operator from the root to the leaves, inference will remain tractable. Generalizing SPFs in this way will increase their expressivity and make it easier to define and learn new SPF architectures. For example, by incorporating sinusoidal or exponential functions into the SPF framework, it may be possible to exactly optimize the non-unary leaf functions used to model protein folding in Chapter 4. This increased flexibility may also alleviate the need for determinism and selectivity in problems that require translating an SPF from one semiring to another.

Decomposability and the semiring properties will remain key to ensuring tractability but, depending on the choice of operators, it may not always be possible to push the marginalization operator past all internal operators. However, by extending the ideas of approximate decomposability defined in Chapter 4, it should be possible to ensure tractable marginalization by approximating the function with a decomposable version. This can be accomplished with local function simplification as in RDIS, auxiliary-variable methods

as in feasible target propagation, variational methods [145], or latent variable discovery methods.

Discriminative SPFs

Alternatively, not all tasks require the ability to marginalize over any subset of variables, as guaranteed by decomposable SPFs. Instead, it is natural to consider learning of and inference in SPFs that only support marginalization over a pre-specified subset of variables. The benefit of reducing the number of marginalization queries that an SPF need support is that decomposability is only necessary for products between variables in the marginalization subset, but not among the remaining variables [47]. This allows for a much larger class of models. Further, variables outside of the marginalization subset can now be combined using arbitrary operators, since the marginalization operator no longer needs to be pushed past these operators to ensure tractability.

Extending decomposable SPFs to allow arbitrary operators and to require decomposability over only a subset of variables defines a spectrum between SPFs and modern deep (feed-forward) neural networks (DNNs), which can now be interpreted as SPFs with arbitrary internal operators (e.g., those used in different layers and for nonlinear activation functions) and with decomposability required over only the class variable. DNN learning is thus an instance of the structured prediction learning problem defined in Section 3.4 with $\max_{y \in \mathcal{Y}}$ as the marginalization operator, where y is the class label and $\mathcal{Y}$ is its domain.

Despite only permitting tractable marginalization over a single variable, DNNs have proved highly successful on a wide variety of tasks in a number of domains, including computer vision, natural language processing, and speech recognition. However, as researchers in these areas move to harder tasks that require models to efficiently support multiple queries, as will be needed for artificial general intelligence, new models will also necessarily move along this spectrum towards decomposable SPFs.

Combining SPFs and DNNs

Models that combine DNNs and decomposable SPFs are an exciting prospect that can be more powerful than either alone. For example, low-level feature recognition in vision is currently dominated by convolutional DNNs, which exploit the translation invariance of objects to learn more accurate object detectors; however, the predictions of DNNs are often poorly calibrated [55], an issue that probabilistic models such as sum-product networks do not suffer from. Sum-product networks that are convolutional and decomposable would enjoy both of these benefits. Further, while DNNs support only a single query, compositional image models such as submodular field grammars support complex queries about both the objects in a scene and their relationships. In Chapter 5, I showed that DNNs can be used to represent low-level features for submodular field grammars. Using DNNs within arbitrary SPFs is a natural extension of this idea. Conversely, SPFs can also be used as layers in DNNs. For example, using a sum-product network as a layer would allow tractable probabilistic inference to be used as a component within a DNN. Alternatively, decomposable min-sum functions (Section 3.5.4), RDIS-like algorithms (Chapter 4), or submodular layers all provide efficient ways to implement optimization layers as proposed by Amos and Kolter [5].

Weight learning in SPFs that are differentiable, such as sum-product networks [47, 114], can be done using backpropagation-based methods, which are also the standard approach for learning DNNs. Models that combine differentiable SPFs and DNNs can thus also be learned effectively with these methods. Otherwise, algorithms for learning models with non-differentiable layers, such as feasible target propagation (Chapter 6), can be used. Structure learning methods for SPFs (e.g., LearnSPF from Section 3.4 or those for sum-product networks [2, 48, 120]) can be used to learn the structures of SPFs that are combined with DNNs, or even for learning the entire architecture since SPFs and DNNs are two ends of the same spectrum. While much work remains to be done in this regard, I believe that learning complex systems of SPFs and DNNs in which each has a different

role, such as object recognition or planning, will enable the development of much more capable models and speed the creation of artificial general intelligence.

BIBLIOGRAPHY

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an SVD-based algorithm. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, pages 32–41, 2015.

[3] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle Adjustment in the Large. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision ECCV 2010*, volume 6312 of *Lecture Notes in Computer Science*, pages 29–42. Springer Berlin Heidelberg, 2010.

[4] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46:325–343, 2000.

[5] Brandon Amos and J. Zico Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 136–145, Sydney, Australia, 2017.

[6] Christian B. Anfinsen. Principles that Govern the Folding of Protein Chains. *Science*, 181(4096):223–230, 1973.

[7] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian inference via backtracking search. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence*, pages 20–28, 2002.

[8] F. Bacchus, S. Dalmao, and T. Pitassi. Solving #SAT and Bayesian inference with backtracking search. *Journal of Artificial Intelligence Research*, 34:391–442, 2009.

[9] Francis Bach and Michael I. Jordan. Thin junction trees. In *Advances in Neural Information Processing Systems*, pages 569–576, 2001.

[10] David Baker. A surprising simplicity to protein folding. *Nature*, 405(6782):39–42, 2000.

[11] J. Barwise. *Handbook of mathematical logic*. Elsevier, 1982.

[12] Roberto J Bayardo Jr. and Joseph Daniel Pehoushek. Counting Models Using Connected Components. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 157–162, 2000.

[13] Yoshua Bengio. How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation. *arXiv preprint*, pages 1–34, 2014.

[14] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint*, pages 1–12, 2013.

[15] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[16] Jeffrey A Bilmes. Deep Mathematical Properties of Submodularity with Applications to Machine Learning. NIPS Conference Tutorial, 2013.

[17] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44:201–236, 1997.

[18] Silvia Bonettini. Inexact block coordinate descent methods with application to non-negative matrix factorization. *IMA Journal of Numerical Analysis*, 31(4):1431–1452, 2011.

[19] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 4, 1996.

[20] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[21] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11): 1222–1239, 2001.

[22] Miguel Á. Carreira-Perpiñán and Weiran Wang. Distributed optimization of deeply nested systems. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2014.

[23] A. Cassioli, D. Di Lorenzo, and M. Sciandrone. On the convergence of inexact block coordinate descent methods for constrained optimization. *European Journal of Operational Research*, 231(2):274–281, 2013.

[24] Ümit Çatalyürek and Cevdet Aykanat. PaToH (Partitioning Tool for Hypergraphs). In David Padua, editor, *Encyclopedia of Parallel Computing*, pages 1479–1487. Springer US, 2011.

[25] V. Chandrasekaran, N. Srebro, and P. Harsha. Complexity of inference in graphical models. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 70–78, 2008.

[26] L. Chang and A. K. Mackworth. A generalization of generalized arc consistency: From constraint satisfaction to constraint-based inference. In *Proceedings of the IJCAI-05 Workshop on Modeling and Solving Problems with Constraints*, 2005.

[27] A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems*, pages 273–280, 2007.

[28] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *Proceedings of the International Conference on Learning Representations*, 2015.

[29] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *ArXiv e-prints*, 2016.

[30] Michael Collins. Discrimative training methods for hidden Markov models: theory and experiments with perceptron algorithms. *Proceedings of the Conference on Empirical Methods in NLP (EMNLP 2002)*, pages 1–8, 2002.

[31] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[32] Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2): 11–34, 2001.

[33] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48: 608–647, 2001.

[34] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126:5–41, 2001.

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

[36] Adnan Darwiche and Mark Hopkins. Using recursive decomposition to construct elimination orders, jointrees, and dtrees. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 180–191. Springer, 2001.

[37] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[38] Leonardo De Moura and Nikolaj Bjørner. Satisfiability Modulo Theories: Introduction and Applications. *Communications of the ACM*, 54(9):69–77, sep 2011.

[39] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

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

[41] Andrew Delong, Lena Gorelick, Olga Veksler, and Yuri Boykov. Minimizing Energies with Hierarchical Costs. *International Journal of Computer Vision*, 100(1):38–58, 2012.

[42] P. Domingos and W. A. Webb. A tractable first-order probabilistic logic. In *Proceedings of the 26th Conference on Artificial Intelligence*, pages 1902–1909, 2012.

[43] P. Domingos, M. Niepert, and D. Lowd, editors. *Proceedings of the ICML-14 Workshop on Learning Tractable Probabilistic Models*. ACM, 2014.

[44] Abram L. Friesen and Pedro Domingos. Recursive Decomposition for Nonconvex Optimization. In Qiang Yang and Michael Woolridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 253–259. AAAI Press, 2015.

[45] 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 (ICML 2016)*, volume 48, 2016.

[46] Abram L. Friesen and Pedro Domingos. Unifying Sum-Product Networks and Submodular Fields. In *Proceedings of the Workshop on Principled Approaches to Deep Learning at ICML*, 2017.

[47] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3239–3247, 2012.

[48] 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. Omnipress, 2013.

[49] Robert Gens and Pedro Domingos. Compositional Kernel Machines. In *ICLR 2017 Workshop Track*, 2017.

[50] Stephen Gould, Richard Fulton, and Daphne Koller. Decomposing a scene into geometric and semantically consistent regions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–8, 2009.

[51] TJ Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems*, pages 31–40. ACM Press, 2007.

[52] D. M. Greig, B.T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(2): 271–279, 1989.

[53] Andreas Griewank and Philippe L. Toint. On the unconstrained optimization of partially separable functions. *Nonlinear Optimization*, 1982:247–265, 1981.

[54] Luigi Grippo and Marco Sciandrone. Globally convergent block-coordinate techniques for unconstrained optimization. *Optimization Methods and Software*, 10(4):587–637, 1999.

[55] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1321–1330, Sydney, Australia, 2017.

[56] Eldon Hansen and G. William Walster. *Global optimization using interval analysis: revised and expanded*, volume 264. CRC Press, 2003.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[59] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient weighted MAX-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[60] Geoffrey E. Hinton. Coursera Lectures: Neural networks for machine learning. Coursera, 2012.

[61] Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

[62] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading MA, 1979.

[63] Jinbo Huang and Adnan Darwiche. The language of search. *Journal of Artificial Intelligence Research*, 29:191–219, 2007.

[64] Itay Hubara, Daniel Soudry, and Ran El-Yaniv. Binarized Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1–17, 2016.

[65] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 448–456, Lille, France, 2015.

[66] Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H. Adelson. Crisp boundary detection using pointwise mutual information. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.

[67] Stasys Jukna. *Boolean Function Complexity*. Springer Berlin Heidelberg, 2012.

[68] Daniel S. Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.

[69] Kalev Kask, Rina Dechter, Javier Larrosa, and Avi Dechter. Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166:165–193, 2005.

[70] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., 2003. ACM.

[71] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *Proceedings of the 5th International Conference on Learning Representations*, pages 1–16, 2016.

[72] Angelika Kimmig, Guy Van Den Broeck, and Luc De Raedt. Algebraic model counting. *arXiv preprint arXiv:1211.4475*, 2012.

[73] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[74] Dan Klein and Christopher D. Manning. A generative constituent-context model for improved grammar induction. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 128–135, 2002.

[75] Vladimir Kolmogorov and Carsten Rother. Minimizing nonsubmodular functions with graph cuts - a review. *IEEE transactions on pattern analysis and machine intelligence*, 29(7):1274–9, 2007.

[76] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2): 147–159, 2004.

[77] Nikos Komodakis, Georgios Tziritas, and Nikos Paragios. Fast, approximately optimal solutions for single and dynamic MRFs. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[78] Andreas Krause and Carlos Guestrin. Beyond convexity: Submodularity in machine learning. ICML Tutorial, 2008.

[79] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[80] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[81] Alex Kulesza and Fernando Pereira. Structured Learning with Approximate Inference. In *Advances in Neural Information Processing Systems*, 2007.

[82] M. Pawan Kumar and Daphne Koller. MAP Estimation of Semi-Metric MRFs via Hierarchical Graph Cuts. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 313–320, 2009.

[83] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13:32–44, 1992.

[84] Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel Vrain. On learning constraint problems. *Proceedings of the International Conference on Tools with Artificial Intelligence*, 1:45–52, 2010.

[85] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.

[86] E. L. Lawler and D. E. Wood. Branch-and-Bound Methods: A Survey. *Operations Research*, 14:699–719, 1966.

[87] Andrew Leaver-Fay, Michael Tyka, Steven M Lewis, Oliver F Lange, James Thompson, Ron Jacak, Kristian Kaufman, P Douglas Renfrew, Colin A Smith, Will Sheffler, and Others. ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. *Methods in Enzymology*, 487:545–574, 2011.

[88] Yann LeCun. Learning Process in an Asymmetric Threshold Network. In E Bienenstock, F Fogelman Soulié, and G Weisbuch, editors, *Disordered Systems and Biological Organization*, pages 233–240. Springer, Berlin, Heidelberg, 1986.

[89] Yann LeCun. *Modèles connexionnistes de l'apprentissage*. PhD thesis, Université de Paris VI, 1987.

[90] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Grégoire Montavon, Geneviève B Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade: Second Edition*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[91] Dong Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 9284, pages 498–515, 2015.

[92] Victor Lempitsky, Carsten Rother, Stefan Roth, and Andrew Blake. Fusion moves for Markov random field optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1392–1405, 2010.

[93] Victor Lempitsky, Andrea Vedaldi, and Andrew Zisserman. A pylon model for semantic segmentation. In *Neural Information Processing Systems*, pages 1–9, 2011.

[94] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training Quantized Nets: A Deeper Understanding. In *Advances in Neural Information Processing Systems*, pages 1–17, 2017.

[95] Darryl D. Lin and Sachin S. Talathi. Fixed Point Quantization of Deep Convolutional Networks. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pages 2849–2858, 2016.

[96] Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Overcoming Challenges in Fixed Point Training of Deep Convolutional Networks. In *Workshop on On-Device Intelligence at ICML*, 2016.

[97] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[98] M. I. A. Lourakis. levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++. http://www.ics.forth.gr/~lourakis/levmar/, 2004.

[99] D.J.C. MacKay. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.

[100] Robert Mateescu and Rina Dechter. The relationship between AND/OR search spaces and variable elimination. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 380–387, 2005.

[101] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, and Hao Wu. Mixed Precision Training. *arXiv preprint*, pages 1–14, 2017.

[102] Willard Miller. *Symmetry groups and their applications*, volume 50. Academic Press, 1972.

[103] Marvin L. Minsky and Seymour Papert. *Perceptrons: an introduction to computational geometry*. The MIT Press, Cambridge, MA, 1969.

[104] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[105] Elchanan Mossel and Sebastien Roch. On the Submodularity of Influence in Social Networks. In *Symposium on Theory of Computing 2007*, pages 128–134, 2007.

[106] Arnold Neumaier, Oleg Shcherbina, Waltraud Huyer, and Tamás Vinkó. A comparison of complete global optimization solvers. *Mathematical Programming*, 103(2): 335–356, 2005.

[107] Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back. *ACM SIGMOD Record*, 42(4):5–16, 2014.

[108] Mathias Niepert and Pedro Domingos. Learning and inference in tractable probabilistic knowledge bases. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, pages 632–641, 2015.

[109] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

[110] A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, pages 615–622. Polytechnic Institute of Brooklyn, 1962.

[111] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kauffmann, San Mateo, CA, 1988.

[112] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *Proceedings of the ICML-14 Workshop on Learning Tractable Probabilistic Models*, 2014.

[113] Slav Petrov. Generative and discriminative latent variable grammars. In *Advances in Neural Information Processing Systems*, 2009.

[114] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 337–346. AUAI Press, 2011.

[115] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing (3rd ed.)*. Cambridge University Press, 2007.

[116] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Proceedings of the 14th European Conference on Computer Vision*, pages 1–17, 2016.

[117] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[118] Emma Rollon, Javier Larrosa, and Rina Dechter. Semiring-based mini-bucket partitioning schemes. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 644–650, 2013.

[119] A Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of the 31st International Conference on Machine Learning*, pages 710–718, 2014.

[120] Amirmohammad Rooshenas and Daniel Lowd. Discriminative structure learning of arithmetic circuits. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pages 1506–1514, 2016.

[121] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[122] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(94): 273–302, 1996.

[123] David E. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learining Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. The MIT Press, 1986.

[124] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[125] Chris Russell, Lubor Ladický, Pushmeet Kohli, and Philip H.S. Torr. Exact and approximate inference in associative hierarchical networks using graph cuts. *The 26th Conference on Uncertainty in Artificial Intelligence*, pages 1–8, 2010.

[126] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining Component Caching and Clause Learning for Effective Model Counting. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[127] Tian Sang, Paul Beame, and Henry Kautz. Performing Bayesian inference by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1, pages 475–482, 2005.

[128] Tian Sang, Paul Beame, and Henry Kautz. A dynamic approach to MPE and weighted MAX-SAT. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 173–179, 2007.

[129] Fabio Schoen. Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, 1(3):207–228, 1991.

[130] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of Gradient-Based Deep Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1–33, 2017.

[131] Abhishek Sharma, Oncel Tuzel, and Ming-Yu Liu. Recursive context propagation network for semantic scene labeling. In *Advances in Neural Information Processing Systems*, pages 2447–2455, 2014.

[132] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. TextonBoost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. *Proceedings European Conference on Computer Vision (ECCV)*, 3951 (Chapter 1):1–15, 2006.

[133] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1): 2–23, 2009.

[134] Amir Shpilka and A Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, 2010.

[135] Parag Singla and Pedro Domingos. Discriminative training of Markov logic networks. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 868–873, 2005.

[136] Richard Socher, Cliff C. Lin, Chris Manning, and Andrew Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, pages 129–136, 2011.

[137] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation Backpropagation: parameter-free training of multilayer neural networks with real and discrete weights. In *Advances in Neural Information Processing Systems*, volume 2, pages 1–9, 2014.

[138] Wei Tang, Gang Hua, and Liang Wang. How to Train a Compact Binary Neural Network with High Accuracy ? In *Proceedings of the 31st Conference on Artificial Intelligence*, pages 2625–2631, 2017.

[139] Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 896–903, 2005.

[140] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training Neural Networks Without Gradients: A Scalable ADMM Approach. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, 2016.

[141] Aleksandar Trifunović. *Parallel algorithms for hypergraph partitioning*. Ph.d., University of London, 2006.

[142] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment – a modern synthesis. In *Vision Algorithms:˜Theory and Practice*, pages 298–372. Springer, 2000.

[143] Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423, 2009.

[144] A. TÂĺorn and A. Zilinskas. *Global Optimization*. Springer-Verlag, 1989.

[145] Martin J. Wainwright and Michael I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends® in Machine Learning*, 1(1âĂŞ2): 1–305, 2007.

[146] W. Austin Webb and Pedro Domingos. Tractable probabilistic knowledge bases with existence uncertainty. In *Proceedings of the UAI-13 International Workshop on Statistical Relational AI*, 2013.

[147] Yair Weiss. Correctness of Local Probability Propagation in Graphical Models with Loops. *Neural Computation*, 12(1):1–41, 2000.

[148] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[149] Nic Wilson. Decision diagrams for the computation of semiring valuations. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 331–336, 2005.

[150] Rodney Winter and Bernard Widrow. MADALINE RULE II: A training algorithm for neural networks. In *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, CA, USA, 1988. IEEE.

[151] Yichao Wu and Yufeng Liu. Robust Truncated Hinge Loss Support Vector Machines. *Journal of the American Statistical Association*, 102(479):974–983, 2007.

[152] Chen Yanover, Talya Meltzer, and Yair Weiss. Linear programming relaxations and belief propagation – an empirical study. *The Journal of Machine Learning Research*, 7: 1887–1907, 2006.

[153] Han Zhao, Mazen Melibari, and Pascal Poupart. On the Relationship between Sum-Product Networks and Bayesian Networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, 2015.

[154] Yibiao Zhao and Song-Chun Zhu. Image parsing via stochastic scene grammar. In *Advances in Neural Information Processing Systems*, pages 1–9, 2011.

[155] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv preprint*, pages 1–14, 2016.

[156] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained Ternary Quantization. In *Proceedings of the 5th International Conference on Learning Representations*, pages 1–10, 2017.

[157] Song-Chun Zhu and David Mumford. A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, 2(4):259–362, 2006.