Building More Expressive Structured Models

by

Yujia Li

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Building More Expressive Structured Models

Yujia Li
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2017

Structured data and structured problems are common in machine learning, and they appear in many applications from computer vision, natural language understanding, information retrieval, computational biology, and many more. Compared to unstructured problems, where the input data is represented as a vector of independent feature values and output is a scalar prediction like a class label or regression value, both the input and output for structured problems may be objects with internal structure, like sequences, grids, trees or general graphs.

Effectively exploiting the structure in the problems can help build efficient prediction models that significantly improve performance. The complexity of the structures requires expressive models that have enough representation capabilities. However, increased model complexity usually leads to increased inference complexity. A key challenge in building more expressive structured models is therefore to balance the model complexity and inference complexity, and explore models that are both expressive enough and have efficient inference.

In this thesis, I present our work in the direction of building more expressive structured models, from developing more expressive structured output models, to semi-supervised learning of structured models, and then structured neural network models.

The first technical part of the thesis describes a model that uses a new family of expressive high order pattern potentials, for which we characterized the theoretical properties and developed efficient inference and learning algorithms. Next we study semi-supervised learning algorithms for structured prediction problems that can help improve prediction performance by using unlabeled data. Motivated by our observation that standard structured models with iterative inference algorithms can be converted to neural networks, we study in particular structured neural network models for structured problems, and propose a new model that can handle prediction problems on graphs.

Discussions about promising future directions are presented at the end of each technical chapter as well as at the end of the thesis.

Acknowledgements

Coming to Canada and studying for a Ph.D. in the machine learning group at University of Toronto is an amazing journey for me. I am deeply grateful to many people that have helped me, supported me and cheered for me during the journey in the past five and a half years.

First I want to thank my supervisor Rich, for always supporting me, giving me guidance when I needed help, and letting me pursue my own research directions when I have ideas. I learned a lot about how to do research from Rich. Being a graduate student, it is always tempting to publish a lot of papers. But I learned from Rich that we should only submit and publish things that we are happy with ourselves, and do research that we can be proud of. This really helped me establish a good attitude about my research work and helped shaping my research directions. In the later stage of my Ph.D., I learned more about how to think at a higher level and keep pushing on the most important directions in problem solving from Rich. Aside from all these, I'm also grateful for all the lunches and home parties that Rich invited me and others from the group to join. And thank you Rich for buying the beer on my last day in Toronto.

I'm also very fortunate and grateful to have Danny Tarlow as a labmate for two years. Danny is always a great role model to me. Danny helped me a lot in my first paper, and it wouldn't be possible without him. I benefitted a lot from Danny's coaching. His way of framing research problems and writing papers has great impact on me. But more importantly I learned a lot from his attitude towards research, Ph.D. work, and life. I was fortunate to have an opportunity to work with Danny again as an intern at Microsoft Research Cambridge on a really fun project, from which I learned how to dream big and ambitious, but also be practical and careful at planning, and take solid steps towards a big goal. I dreamed of achieving a fraction of what Danny did during his Ph.D., I hope I have partially achieved that goal by now.

I also want to thank Kevin for helping me in a number of projects, he is always encouraging, and always has ways and ideas to make things better. Kevin and I entered the Ph.D. program in the same year, but he has way more experience than me. I have learned a lot from our collaborations.

I thank my great officemates, Charlie for always sharing his enthusiasm, Maks for being a model for hard working, James for providing sharp insights on all kinds of things, Jamie and Ryan for the helpful discussions and daily chats that made the lab a pleasant place, Jake for sharing all the fun with me during the years and inspiring me to take on running as a good habit, Renjie and Mengye for always giving me pressure to be better.

I thank all my collaborators. I thank Wenjie for working tirelessly on our project and being a good friend, Raquel for thought-provoking discussions and being a great teacher, Marc for always being helpful and all the fun he shared with us, Christos for his hard work, Kaisheng for being a great mentor, Geoff Zweig for sharing his inspiring insights in the intern project. At the end of my Ph.D. I have got to interact with Alex Schwing more and we started working on a fun project together, I learned a lot about optimization from Alex and his way of doing research has impacted me greatly. I also thank Kamyar and Jackson for the hard work they put in our collaboration.

I thank David and Russ for serving on my supervisory committee, providing guidance to me throughout my Ph.D.. I also thank Raquel and Andrew for agreeing to spend time reading my thesis and providing helpful feedback. I thank all other members at the UofT machine learning group and many other people that I had the fortune to interact with, Shenlong, Jian, Eleni, Kelvin, Roger, Jasper, Tijmen,

Chris, Tony, Jimmy, Ilya, Ali, Kaustav, Namdar, Min, Yanshuai, Mohammad, Abdel-Rahman, Tom, Fartash, Nitish, Navdeep, George, Elman, Vlad, Alex Krizhevsky, Alex Graves, Sanja, Geoff Hinton, Ladislav, Alex Gaunt, Sid, Avner, Ritwik, Yasha, Radford, Brendan, Jeroen, Emily, Shane, Chi and many more that I didn't name. I thank Relu for providing excellent systems support and making our computers running and up to date, Luna for providing excellent admin support.

I thank Canada and the city of Toronto for being such amazing and welcoming places.

I'm also grateful that my undergrad advisor Xiaoming Jin introduced machine learning to me and get me started in research. I thank Xuetao and Lianghao for the work they put in the IJCAI paper and also being good friends. I thank Xiaochen and Lei for the happy memories in our Tsinghua lab as well.

I thank my parents and all members of my family for the support along the journey.

At last, I thank my wife, then girlfriend when I started the Ph.D., and my best friend. Thank you for always cheering for every small progress I made during the years, thank you for sharing everything in life with me, and thank you for making this journey more enjoyable and unforgettable.

Contents

Chapter 1

Introduction

This thesis focuses on building expressive *structured* models. Many machine learning problems in a wide range of applications have rich structures. These structures are inherent properties of data, and can be utilized to build models that take such structures into account. Classical examples of such structures include spatial structure in visual data and temporal structure in sequence data. A major part of the work in this thesis is on *structured output* models [93, 161, 128], which are supervised learning models that predict structured objects, rather than scalar values like discrete classification predictions, or real valued regression predictions. In this thesis, the term *structured models* is also not limited to only structured output models, but also includes structured input models, which model structured input data and make a structured or unstructured prediction.

In general, the term *structure* is used to refer to the inherent relationship between input data entities or output prediction entities. Structures exist in most problems and data, but some models make assumptions that ignore these structures. Structured models, on the other hand, make use of these structures to improve task performance.

1.1 Examples of Structured Data and Problems

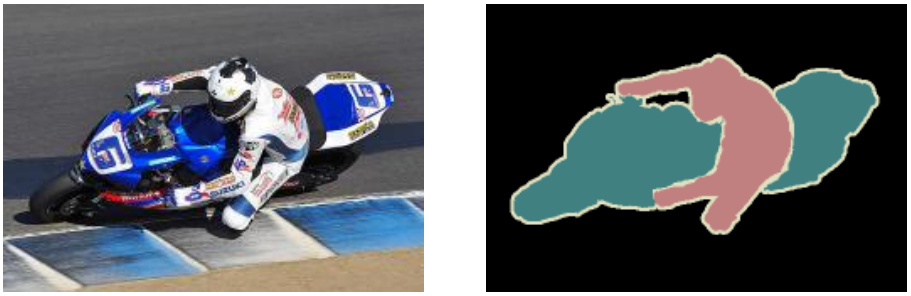
We start by describing the *unstructured* problem setup, where no structure is considered in the input and output. Then we move on to present a few examples of structured data and problems, to illustrate some possible application scenarios and compare with the unstructured setup. The examples presented here are by no means exhaustive, but selected to be representative.

The Unstructured Problem Setup For a long period of time, the most common machine learning problem setup looked like this: the input data is represented as feature vectors $\mathbf{x} \in \mathbb{R}^n$, and the output of a model is a scalar value y, either a regression score $y \in \mathbb{R}$ or a classification decision $y \in \{1, ..., K\}$.

This setup can handle problems where both the input and the output are not structured. On the input side, this setup separates feature extraction and learning, and the learning algorithm assumes that each dimension of the extracted features is independent from each other; thus no structure in the input space is considered by the model. On the output side, only scalar value predictions are made, therefore many example problems presented below could not be handled in this setup.

Input:	The	grand	jury	commented	on	a	number	of	other	topics	.
Output:	DT	JJ	NN	VBD	IN	DT	NN	IN	JJ	NNS	.

Figure 1.1: POS tagging problem example. Tag code: DT - determiner, JJ - adjective, NN - noun, singular or mass, VBD - verb, past tense, IN - preposition, NNS - noun, plural. This example comes from the Penn Treebank [115].



Input Output

Figure 1.2: Semantic image segmentation example. Label colors: black - background, cyan - motorbike, magenta - person, grey - unlabelled. This example comes from the PASCAL VOC Challenge [37].

Sequence Prediction Sequence prediction is the task of predicting a sequence of outputs for a given input. Such tasks are common in language understanding, as language data are naturally sequences of words or tokens.

One classic example is the part-of-speech (POS) tagging task. In this task, the input data is a sequence of words, and the output is a sequence of POS tags, one for each word in the input sequence. One example input and output pair is illustrated in Fig. 1.1.

In this problem, the input sequence is a structured object composed of a sequence of words, where the combination of the words follow a rich set of structures, forming phrases and clauses and following grammars. The output sequence is also structured, as the sequence of POS tags should be consistent with the grammar of the language. Utilizing these structures can greatly improve performance of the models developed for this task [93].

Another example of the sequence prediction task is language generation [153, 74]. Here a sequence of words, usually a sentence, is generated as output in response to some input, which can be another sequence, as in question answering or conversation systems, or images as in image to text generation. In this problem the input may sometimes be unstructured and represented by a feature vector, but the output is still structured.

Semantic Image Segmentation Semantic image segmentation refers to the task of assigning a semantic class label to every pixel in an image, effectively grouping pixels into segments belonging to different semantic classes. The spatial arrangement of the pixels in an image is an important structure in this problem. The input image has this spatial structure, as do the output labelings. One example input-output pair is shown in Fig. 1.2.

It is possible to treat this problem as an unstructured prediction problem, where feature vectors are extracted for each pixel, and then a classifier is trained on top of the feature vectors to make predictions for each pixel independently. However, incorporating the spatial structure into the prediction process can result in significant improvement of performance.

Other dense prediction tasks where one prediction has to be made for each pixel in an image, include depth estimation and optical flow estimation. These problems share a very similar formulation as the segmentation problem.

A large part of this thesis uses the segmentation problem as the target application, as the problem is challenging and there are rich structures in the input and output.

Ranking Ranking is an example where the input data is only loosely structured, but the output is structured [108, 18]. The input to a ranking system is a set of (query, document) pairs, usually each represented as a feature vector, and there is no obvious structure between different documents. The output is a ranking of all the documents for a given query, which has to put the documents in order of descending relevance to the query. Usually, an error made at the top of the ranking is much more undesirable than an error made at the bottom of the ranking. The structure of the output is therefore very important.

Predictions on Structured Input Data There are also other examples that use structured input data, and produce an unstructured prediction. Making use of the structure in input data, and developing structured models for these problems is also the key to get good performance.

Image classification is one example that can be treated as an unstructured problem, by using feature vectors and follow the common machine learning paradigm. However, making use of the spatial structure of image data can greatly improve performance, as demonstrated by the rapid improvement of the state-of-the-art on image classification by developing convolutional neural networks [87, 144, 56]. The key to the success of these models is the use of the special convolution architecture designed for visual data.

Another example is predictions on graphs. In such problems the input data is a graph structure, represented using graph nodes and edges, and the output maybe to classify if the graph contain some special property. These problems occur naturally in examples such as the computational analysis of molecule structure [34], reasoning with knowledge graphs [106], etc..

1.2 Structured Models

The straightforward approach to solving structured problems is to ignore the inherent structure and treat the data as collections of independent individual pieces of information, and then reduce the problem to well-studied classification or regression problems under the independent identically distributed (i.i.d.) assumption. One classic approach following this idea is the "bag of words" model for modeling text data, which completely ignores the word ordering in text, and simply uses word counts in a document as its representation for machine learning tasks. While this model has some success in problems like text classification, it is not adequate for even the simplest problems that require some understanding of word ordering, not to say the more challenging language understanding tasks. This motivates the need for developing structured models that can exploit the structures in data.

Structured output models have been proposed to model the structure in the outputs. One form of these models defines a scoring function $F(\mathbf{x}, \mathbf{y})$ for a pair of input $\mathbf{x}$ and output $\mathbf{y}$, which is usually defined in a way to exploit the structure in $\mathbf{y}$, and then makes a prediction for a certain input $\mathbf{x}$ by optimizing the scoring function to find the optimal output $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} F(\mathbf{x}, \mathbf{y})$. The structured SVM model [161] follows this approach. Another form of structured output model tries to model the conditional

distribution $p(\mathbf{y}|\mathbf{x})$ instead and makes predictions by doing probabilistic inference under this model. The structure of $\mathbf{y}$ is reflected in the structure of the probability distribution, and usually represented with probabilistic graphical models. The Conditional Random Field (CRF) [93] is such a probabilistic model. These two approaches can also be unified, as the conditional distribution $p(\mathbf{y}|\mathbf{x})$ can usually be defined via a scoring function through the Gibbs distribution $p(\mathbf{y}|\mathbf{x}) = \exp(F(\mathbf{x}, \mathbf{y}))/\sum_{\mathbf{y}'} \exp(F(\mathbf{x}, \mathbf{y}'))$.

Structured input models on the other hand focus on learning good representations for structured input data, rather than using hand engineered features. These models are usually neural network models. Examples of such models include convolutional neural networks widely used in computer vision, and recurrent neural networks for learning on sequential data. In this thesis, we also introduce graph neural networks that can learn representations for abstract graph data. These models are developed to efficiently exploit the structure in the input data, and can learn good representations for structured or unstructured prediction tasks.

These structured models have achieved great success in the past, as can be seen from, for example, the rise of convolutional neural networks in computer vision and also the popularity of graphical models for representing structures and relations in data. The structured output and input models can also be combined to work jointly, and do predictions from end to end. The neural network models can also be used directly to predict structured output $\mathbf{y}$ from input $\mathbf{x}$ through a network that does the computation $\mathbf{y} = F(\mathbf{x})$.

1.3 Challenges in Building Structured Models

Building expressive structured models has its own unique challenges.

The biggest challenge is the difficulty in inference, or the process of making a prediction, when the model is complicated. Due to the structure in the output space, making a structured prediction should consider all the correlations between different components of an output object. When such correlations are high-order, i.e. involving many (more than 2) components, making the optimal prediction becomes hard, as the possible number of types of interaction between components grows exponentially. On the other hand, even for a model with only pairwise interactions between components, for most problem instances the deterministic inference problem (finding the argmax of the scoring function) is still intractable [14], and probabilistic inference can only be done approximately; the performance is not guaranteed to be good; and sometimes an inference algorithm may even fail to converge to a solution. The key challenge of building more expressive structured models is therefore to build models that can better handle a larger class of structures, but at the same time allow efficient and accurate inference, which is very challenging. This challenge is unique to structured prediction models, as in simpler tasks like classification or regression there is no structure in the output space because only a single scalar value is predicted.

The difficulty of inference can also cause trouble for learning, as learning a structured model usually uses the inference algorithm as a subroutine. Since the inference is usually hard, and in most cases we can only get approximate inference results, making sure the learning algorithm still works and gets the right training signal is therefore very important. Learning algorithms should be compatible with the inference algorithm used to make predictions.

Another challenge for learning expressive structured models is the requirement of large amounts of structured training data. Obtaining such structured data, especially structured output labels, requires

much more effort and expertise for labelling than unstructured data. Efficiently utilizing the structure of data may alleviate the problem of requiring large training data set. Utilizing unlabelled data may also help.

Structured models also share some challenges with other models. For example, when using a nonlinear model like a neural network to model structured input data, or directly using one as prediction models, training is a challenge because of the nonlinearity and nonconvexity of the model. The interpretability of such models is also a challenge, as it is hard to explain why neural network models work the way they do and practically the models that are more easily explained and understood are always more preferable if everything else is equal.

1.4 Contributions

In this thesis, I present a collection of my work (done with my advisor and collaborators) with the theme on building more expressive structured models. The contributions are summarized as follows:

- We discovered the equivalence between a powerful family of structured models based on the so-called "pattern potentials" and restricted Boltzmann machines (RBMs). Then we developed a new family of structured models with compositional high order pattern potentials (CHOPPs) that unifies these two models, and developed efficient inference and learning algorithms for it. These learned CHOPPs greatly expands the power of structured models and can model more expressive structures in the output space than before. This part is presented in Chapter 3.

- We propose a method for doing semi-supervised learning for structured output models that can efficiently utilize unlabelled data to improve supervised learning of such models. The key to the success of semi-supervised learning methods is to control how the information from labelled data can be propagated to unlabelled data. For structured output problems this is particularly challenging as it is harder to model the interactions between examples because of the complex structures in the input and output spaces. We developed the high-order regularization method for modeling such interactions, and we found a connection between our proposed approach and another popular approach for semi-supervised learning called posterior regularization. This part is presented in Chapter 4.

- In Chapter 5, we show that many standard structured output models using an iterative inference algorithm can be converted into a neural network model with a special structure, therefore using neural network models in place of the more established structured output models is a viable alternative for solving structured output problems. Further more, using neural network models allows us to have full control of the computation process and therefore giving us a lot of freedom to design good architectures without worrying about intractable inference.

- We propose a neural network model particularly designed for graph-structured data that can learn representations of graph data, make structured or unstructured predictions on graphs, and even make sequences of such predictions. This model achieves excellent performance on a range of prediction problems on graphs. This part is presented in Chapter 6.

1.4.1 Relationship to Published Work

All of the main technical chapters presented in this thesis have been peer-reviewed and published in machine learning related conferences (Chapter 5 is based on a workshop paper at ICML, but was also peer-reviewed). Below I list the related published papers by chapter:

Chapter 3: Yujia Li, Daniel Tarlow and Richard Zemel. *Exploring Compositional High Order Pattern Potentials for Structured Output Learning.* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.

Chapter 4: Yujia Li and Richard Zemel. *High Order Regularization for Semi-Supervised Learning of Structured Output Problems.* International Conference on Machine Learning (ICML), 2014.

Chapter 5: Yujia Li and Richard Zemel. *Mean Field Networks.* ICML workshop on Learning Tractable Probabilistic Models, 2014.

Chapter 6: Yujia Li, Daniel Tarlow, Marc Brockschmidt and Richard Zemel. *Gated Graph Sequence Neural Networks.* International Conference on Learning Representations (ICLR), 2016.

During my Ph.D. study, I have also worked on a few other directions that are only loosely related to the thesis topic, and therefore not presented in the thesis, including: generative models [104], fair and unbiased feature learning [103, 110], study of convolutional neural network receptive field properties [111], and applications in wearable devices [105] and recommendation on social networks [31].

1.5 Structure of this Thesis

In the rest of this thesis, I present background and some related work in Chapter 2, and then present the 4 main technical chapters, Chapters 3-6, after which I conclude with a summary and discuss some future research directions in Chapter 7.

Chapter 2

Background

In this chapter, I present some important background for the main chapters of this thesis. I start with notations and terminology; then discuss established structured output models, and related inference and learning methods; next I introduce neural network models and describe a few models that can handle structured input data.

2.1 Notations, Terminology and Standard Structured Output Models

In supervised structured learning problems, we have an *input domain* $\mathcal{X}$ and an *output domain* $\mathcal{Y}$, where both $\mathcal{X}$ and $\mathcal{Y}$ can be sets of structured objects. A structured model learns a mapping from $\mathcal{X}$ to $\mathcal{Y}$. For structured output problems there are two popular classes of models: one based on scoring functions, and the other one based on modeling the conditional distributions.

2.1.1 Scoring Function Based Models

In the standard structured output problem setup based on *scoring functions*, a model specifies a function $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{Y} \mapsto \mathbb{R}$ for each $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ that measures the compatibility of the output $\mathbf{y}$ with the input $\mathbf{x}$. The model is parameterized with a parameter vector $\boldsymbol{\theta}$. A prediction is obtained by running an *inference* algorithm that solves the optimization problem

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) \tag{2.1}$$

for a given input $\mathbf{x}$, which finds the optimal $\mathbf{y}^*$ for that $\mathbf{x}$. In this thesis, we use boldface letters to represent vectors or structured objects in general, and use normal letters to represent scalar values, scalar components of a vector or unstructured objects.

As a concrete example, consider the semantic image segmentation problem. The input domain $\mathcal{X} = \mathbb{R}^{H \times W}$ contains all images [1] of size $H \times W$, the output domain $\mathcal{Y} = \{1, 2, ..., K\}^{H \times W}$ contains all

[1] For clarity, we assumed all images have the same size. But in general structured models do not require all inputs to have the same size, and the presentation here do not rely on this assumption. The key is to make the model parameterization not dependent on the size of the inputs or outputs. The presented formulation also only considers a single channel for the input image, but extending it to color images is trivial, by extending the domain $\mathbb{R}$ to $\mathbb{R}^3$ for each pixel.

segmentations that assign one of the K semantic class labels in $\{1, 2, ..., K\}$ to each pixel in the image. Both the input domain and the output domain have local structures where nearby pixels are likely to have similar appearance and labelings.

A common scoring function for structured output problems is

$$F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) = \sum_i f_i(y_i, \mathbf{x}, \boldsymbol{\theta}_u) + \sum_{i,j} f_{ij}(y_i, y_j, \mathbf{x}, \boldsymbol{\theta}_p) \tag{2.2}$$

which decomposes the scoring function for a pair of $(\mathbf{x}, \mathbf{y})$ into terms that score each individual y_i's and pairs of (y_i, y_j)'s. Here the first sum over i goes over all components in an output object, and the second sum over i, j goes over all pairs of components (i, j) that has a connection in an output object. The connection structure is defined by a graph $(\mathcal{V}, \mathcal{E})$, where each node $i \in \mathcal{V}$ in the graph corresponds to one component of the output object, and the edges $(i, j) \in \mathcal{E}$ connect nodes that are correlated. The design of this graph is problem dependent, and usually is made to efficiently exploit the inherent structure in the problem. For semantic segmentation, the graph structure comes from the spatial locality in this problem and is usually modeled as a grid, with each pixel being one node and edges connecting pixels next to each other in the image in 4 or 8 directions.

Here the scoring functions for individual y_i's, $f_i(y_i, \mathbf{x}, \boldsymbol{\theta}_u)$, are called *unary potentials*, and scoring functions for pairs $f_{ij}(y_i, y_j, \mathbf{x}, \boldsymbol{\theta}_p)$ are called *pairwise potentials*. $\boldsymbol{\theta}_u$ and $\boldsymbol{\theta}_p$ are components of the parameter vector $\boldsymbol{\theta}$ for unary and pairwise potentials respectively. The dependence on $\mathbf{x}$ and $\boldsymbol{\theta}$ will be dropped in the potential functions wherever possible if it is not ambiguous, and we use $f_i(y_i)$ and $f_{ij}(y_i, y_j)$ to represent unary and pairwise potentials.

The number of variables involved in a potential function is called the *order* of the potential function. In general the term order measures the number of variables involved in some interaction. The example scoring function in Eq. 2.2 only considers up to second order potential functions and it is therefore limited to only modeling second order interactions between variables. To increase the expressiveness of the model, we can extend the scoring function to include *high order potentials*, which are scoring functions on sets of three or more variables. However, using high order potentials presents some special challenges for inference and learning, as we will see later in the following chapters.

2.1.2 Probabilistic Models

Probabilistic models use a conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$ to model the input to output mapping. Note that this distribution is parameterized by $\boldsymbol{\theta}$, but in this section we make the dependence on $\boldsymbol{\theta}$ implicit for clarity. Unlike the scoring function based approaches, modeling conditional distributions also provides some uncertainty measure, which can be useful in many applications. The structure of the outputs can be exploited and modeled through the structure of the distribution. The study of such structured distributions is the focus of research in the area of probabilistic graphical models [170, 78], as the structure of the distributions is usually specified through graphs. There are two main types of probabilistic models, directed models and undirected models. We describe the formulations for these two models on modeling the conditional distribution $p(\mathbf{y}|\mathbf{x})$, to be consistent with the presentation of conditional models. But we note that these two types of models can also be used to model arbitrary distribution $p(\mathbf{y})$ in general, by simply dropping the dependence on $\mathbf{x}$.

Directed models specify the distribution $p(\mathbf{y}|\mathbf{x})$ through the chain rule

$$p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}_{A_1}|\mathbf{x}) \prod_{i=2}^{n} p(\mathbf{y}_{A_i}|\mathbf{y}_{\cup_{i'<i} A_{i'}}, \mathbf{x}), \tag{2.3}$$

where A_i are subsets of the component indices of $\mathbf{y}$, $\mathbf{y}_A$ restricts $\mathbf{y}$ to a subset of components indexed by A, $A_i \cap A_{i'} = \varnothing$ for all $i \neq i'$, and $\cup_{i=1}^{n} A_i$ covers all the components of $\mathbf{y}$. To fully define this distribution, each component distribution $p(\mathbf{y}_{A_i}|\mathbf{x})$ and $p(\mathbf{y}_{A_i}|\mathbf{y}_{\cup_{i'<i}}, \mathbf{x})$ are modeled using parametric family of distributions or as probability tables. This model is called a directed model because through the definition of this distribution a certain ordering of the components A_i must be specified, and the dependency structure can be represented as a directed graph.

The structure of the output space is modeled by the structure of the distribution. Such structures are defined through the conditional independence properties of the distribution. Two subsets of components in $\mathbf{y}$, $\mathbf{y}_A$ and $\mathbf{y}_B$, are said to be independent conditioned on a third set $\mathbf{y}_C$, if $p(\mathbf{y}_A, \mathbf{y}_B|\mathbf{y}_C, \mathbf{x}) = p(\mathbf{y}_A|\mathbf{y}_C, \mathbf{x})p(\mathbf{y}_B|\mathbf{y}_C, \mathbf{x})$ holds for all $\mathbf{x}$ and any instantiation of $\mathbf{y}_A, \mathbf{y}_B$ and $\mathbf{y}_C$ for the fixed sets A, B, C. This conditional independence property implies that the subsets of components indexed by A and B can only interact through components indexed by C. If such structures exist in a problem, then it is convenient to model these structures using the conditional independence properties.

The conditional independence property definition is also equivalent to $p(\mathbf{y}_A|\mathbf{y}_B, \mathbf{y}_C, \mathbf{x}) = p(\mathbf{y}_A|\mathbf{y}_C, \mathbf{x})$. Therefore specifying the conditional independence properties is equivalent to simplifying the components in the joint distribution definition, by leaving out parts in the conditions for each component.

Undirected models specify the distribution $p(\mathbf{y}|\mathbf{x})$ through a set of factors

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} \prod_{C} \psi_C(\mathbf{y}_C, \mathbf{x}). \tag{2.4}$$

C's are sets of indices, or sets of nodes as in a graphical representation; they are usually also called *cliques*. $\psi_C(\mathbf{y}_C, \mathbf{x}) > 0$ are functions defined on subsets $\mathbf{y}_C$ of $\mathbf{y}$, called *factors*. These factors are usually parameterized by some parameter vector. The normalizing constant

$$Z = \sum_{\mathbf{y}} \prod_{C} \psi_C(\mathbf{y}_C, \mathbf{x}) \tag{2.5}$$

is also called the *partition function*. There are no particular node orderings in undirected models. Undirected models are also called Markov Random Fields (MRFs) in general, when used to model conditional distributions as in this thesis, they are called Conditional Random Fields (CRFs) [93].

It is common to use a *factor graph* [88] to represent undirected models. An example factor graph is shown in Fig. 2.1. In factor graphs, circle nodes corresponds to variables, and squared nodes correspond to factors, each node is connected to the factors it is involved in. Note that each node can involve in multiple different factors. The factor graph representation makes it easy to see the structure of the model and provides a convenient representation for developing message passing algorithms used for inference, which will be discussed later.

Undirected models can also model conditional independence properties. The definition of conditional independence is the same as in directed models, and these properties can be inferred clearly from the factor graph representation. More specifically, it can be shown that in a factor graph, if all paths from

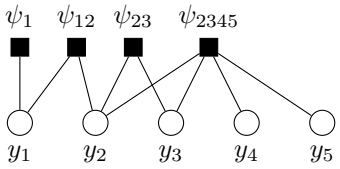


Figure 2.1: Factor graph example for distribution $p(y_1, y_2, y_3, y_4, y_5|\mathbf{x}) \propto \psi_1(y_1, \mathbf{x})\psi_{12}(y_1, y_2, \mathbf{x})$ $\psi_{23}(y_2, y_3, \mathbf{x})\psi_{2345}(y_2, y_3, y_4, y_5, \mathbf{x})$.

any node in set $\mathbf{y}_A$ to any node in set $\mathbf{y}_B$ pass through at least one node in set $\mathbf{y}_C$, then $\mathbf{y}_A$ and $\mathbf{y}_B$ are conditionally independent given $\mathbf{y}_C$.

The discussion of structured probabilistic models in this thesis focuses on undirected models.

2.1.3 Connection between Scoring Function Models and Probabilistic Models

Models based on scoring functions can be naturally converted into undirected probabilistic models via the Gibbs distribution definition

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\exp(F(\mathbf{y}, \mathbf{x})), \tag{2.6}$$

where $Z = \sum_{\mathbf{y}} \exp(F(\mathbf{y}, \mathbf{x}))$.

Alternatively, the factors ψ_C in an undirected probabilistic model can be defined via potential functions f_C as

$$\psi_C(\mathbf{y}_C, \mathbf{x}) = \exp(f_C(\mathbf{y}_C, \mathbf{x})). \tag{2.7}$$

Therefore the distribution specified by an undirected model can be equivalently rewritten as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_C \psi_C(\mathbf{y}_C, \mathbf{x}) = \frac{1}{Z}\exp\left(\sum_C f_C(\mathbf{y}_C, \mathbf{x})\right), \tag{2.8}$$

which implies a scoring function

$$F(\mathbf{y}, \mathbf{x}) = \sum_C f_C(\mathbf{y}_C, \mathbf{x}). \tag{2.9}$$

Because of this connection, undirected models are usually also defined through scoring functions. Factor graphs can also be used to represent the structure of scoring functions. Essentially undirected models and scoring function based models are the same family of models, and a lot of the discussion of them can be unified by studying different operations on the scoring functions. The only difference is that sometimes the probabilistic interpretation can provide extra benefits with respect to model uncertainty.

2.2 Defining the Scoring Functions

Defining the scoring function $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})$ is problem dependent. For a long period of time *log-linear models* where

$$F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) \tag{2.10}$$

is the dominant model for structured output learning. In such models, the F function is a linear function of parameters $\boldsymbol{\theta}$, and $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ is a set of fixed joint feature functions defined on $\mathcal{X} \times \mathcal{Y}$, which can be nonlinear in $\mathbf{x}$ and $\mathbf{y}$.

For pairwise scoring functions as in Eq. 2.2, a common choice for the unary potentials is to use a classifier which specifies the preferences for a particular y_i independent of all other $y_{i'}$'s. The pairwise potentials are usually functions that measure how compatible two nearby labels are to each other. For semantic segmentation problems, the Potts model [134, 13] for pairwise potentials is popular, which is defined as

$$f_{ij}(y_i, y_j, \mathbf{x}, \theta_p) = \theta_p \phi_{ij}(y_i, y_j, \mathbf{x}) = -\theta_p w_{ij}(\mathbf{x}) \mathbf{I}[y_i = y_j]. \tag{2.11}$$

Here $w_{ij}(\mathbf{x})$ measures how similar pixel i and pixel j are based on appearance in $\mathbf{x}$, and θ_p is simply a weighting parameter for the pairwise term.

Nonlinear models like neural networks can also be used as the scoring function. In such models the function F is a nonlinear function of $\boldsymbol{\theta}$. The nonlinearity causes troubles for training, but basic techniques like gradient-based optimization can still be used for training.

In practice, the feature functions in log-linear models can also be nonlinear models, for example unary potentials modeled with neural networks. But in log-linear models such feature functions are trained offline and then fixed, separate from the process of learning $\boldsymbol{\theta}$. The nonlinear model perspective jointly learns all the parameters in the model, and therefore can achieve significant performance improvement [101, 25].

2.3 Inference

Inference in structured output models is the process of finding a proper $\mathbf{y}$ as the output prediction for an input $\mathbf{x}$ under a given model with parameters $\boldsymbol{\theta}$ fixed. The maximum-a-posteriori (MAP) inference problem finds the optimal $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} F(\mathbf{x}, \mathbf{y})$ for a given $\mathbf{x}$, this is equivalent to finding the $\mathbf{y}$ that has the highest posterior probability under probabilistic models for $p(\mathbf{y}|\mathbf{x})$. Besides MAP inference, it is also possible to do probabilistic inference for probabilistic models that can provide some extra uncertainty measure beyond the MAP solution.

2.3.1 MAP Inference

MAP inference finds the $\mathbf{y}^*$ that maximizes the scoring function. This is usually a combinatorial optimization problem for discrete output space. All the problems considered in this thesis fall into this category, we only touch very lightly on continuous output space or mixed output space in this thesis, the study of these cases is beyond the scope of this thesis.

In unstructured learning problems, making predictions is usually a trivial process after the model is learned. For classification problems, we only need to enumerate all the classes and check which one has the highest score. For regression problems, the output of the model is the thing being computed.

The inference problem in structured learning problems is one of the important aspects that distinguish them from unstructured learning problems. Due to the complexity of the output domain, enumerating all possible output objects in the domain is usually intractable. In the image segmentation task, for example, the size of the output domain is exponential in the number of pixels in an image, therefore enumeration is intractable even for the tiniest images with a few hundreds of pixels. This makes the

inference an especially challenging problem for structured output models.

Efficiently solving the inference problem for arbitrary scoring functions and output domains is in general intractable. However, for certain special classes of scoring functions and output domains, efficient inference algorithms can be developed by exploiting the structure of the scoring functions.

Chain models are one of the simplest family of tractable models. The scoring function in a chain model has the following form

$$F(\mathbf{y}) = f_1(y_1) + \sum_{i=2}^{T} \left[f_{i-1,i}(y_{i-1}, y_i) + f_i(y_i) \right]. \tag{2.12}$$

Here we ignored the dependence on $\mathbf{x}$ and $\boldsymbol{\theta}$ from the scoring functions for convenience, but in practice all the potentials may dependent on $\mathbf{x}$ and $\boldsymbol{\theta}$. This model contains unary and pairwise potentials, and all the variables connect to each other through a chain where y_i is connected to y_{i-1} and y_{i+1}. T is the number of variables in $\mathbf{y}$.

The optimal $\mathbf{y}^*$ and the maximum score achievable can be computed through dynamic programming for this model. The key is to use auxiliary functions

$$m_t(y_t) = \max_{\mathbf{y}_{1:t-1}} \; f_1(y_1) + \sum_{i=2}^{t} \left[f_{i-1,i}(y_{i-1}, y_i) + f_i(y_i) \right], \tag{2.13}$$

which can be computed efficiently through the recurrence

$$\begin{aligned} m_t(y_t) &= \max_{\mathbf{y}_{1:t-1}} \left\{ f_1(y_1) + \sum_{i=2}^{t-1} \left[f_{i-1,i}(y_{i-1}, y_i) + f_i(y_i) \right] + f_{t-1,t}(y_{t-1}, y_t) + f_t(y_t) \right\} \\ &= f_t(y_t) + \max_{y_{t-1}} \left\{ f_{t-1,t}(y_{t-1}, y_t) + \max_{\mathbf{y}_{1:t-2}} \left[f_1(y_1) + \sum_{i=2}^{t-2} \left[f_{i-1,i}(y_{i-1}, y_i) + f_i(y_i) \right] \right] \right\} \\ &= f_t(y_t) + \max_{y_{t-1}} \left[f_{t-1,t}(y_{t-1}, y_t) + m_{t-1}(y_{t-1}) \right]. \end{aligned} \tag{2.14}$$

Once these auxiliary functions are computed, the maximum score can be computed easily as $\max_{\mathbf{y}} F(\mathbf{y}) = \max_y m_T(y)$. The optimal solution $\mathbf{y}^*$ can be recovered by tracing back the optimal sequence of $\mathbf{y}$ from the very end. The computational complexity of this process is only polynomial in T. Exploiting the chain structure and utilizing the factorization of the scoring function (the second and third line of the derivation of the recurrence) is the key to reducing the complexity of the inference problem. This dynamic programming approach can also be extended to tree-structured models. The auxiliary functions m so derived are called *messages*. Such algorithms that passes some message functions on a graphical model is in general called *message passing* algorithms, and are widely used for both MAP inference and probabilistic inference described later.

Pairwise models with submodular pairwise potentials are another important family of models with tractable MAP inference solutions. These models are also pairwise models but are not restricted to the chain or tree structures. In particular, in a pairwise model, if the output domain $\mathcal{Y} = \{0, 1\}^N$ is

a set of binary labelings, and the pairwise potentials satisfy the constraint that

$$f_{ij}(y_i, y_j) = 0 \quad \Leftrightarrow \quad y_i = y_j \tag{2.15}$$

$$f_{ij}(y_i, y_j) \quad \leq \quad 0 \tag{2.16}$$

for any i, j and y_i, y_j, then maximizing the scoring function can be converted into an equivalent maximum-flow / minimum-cut problem [13], which has efficient polynomial time algorithms.

This class of scoring functions favor neighboring output components to take the same label, which in many cases is a good prior. Pairwise potentials satisfying these constraints are a special case of the class of *submodular* pairwise potentials. Efficient inference algorithms exist for many submodular scoring functions [85].

However, in most cases, efficiently finding the exact optimal y^* that maximizes the scoring function even for a pairwise model is still intractable and in many cases provably hard [13], therefore we have to live with approximate solutions. In [13], a graph-cut based move-making algorithm is proposed to find approximate solutions when $\mathcal{Y} = \{1, 2, ..., K\}^N$ is not limited to a set of binary labelings. The algorithm is efficient and has been the driving force behind the progress in semantic segmentation research for a long time [76, 44, 20, 21].

High order models with cardinality potentials are an example of tractable models with high order potentials. A cardinality potential [51] on a set of binary variables $\mathbf{y} \in \{0, 1\}^N$ is defined as a function of the number of y_i's being 1, $f_{\text{card}}\left(\sum_i y_i\right)$. Here the function f_{card} can be any arbitrary function defined on integers. Cardinality potentials can therefore be used to specify the desired amount of each type labels in a typical output.

The MAP inference problem for a model with unary potentials and a cardinality potential can be solved efficiently through sorting. To see this, we can first convert the scoring function into the following

$$F(\mathbf{y}) = \sum_i f_i(y_i) + f_{\text{card}}\left(\sum_i y_i\right) = \sum_i y_i \left[f_i(1) - f_i(0)\right] + f_{\text{card}}\left(\sum_i y_i\right) + \sum_i f_i(0). \tag{2.17}$$

Here $\sum_i f_i(0) = C$ is a constant and therefore irrelevant for inference. The first sum is a sum of $f_i(1) - f_i(0)$ terms for y_i's being 1. To find the maximum score, we can (1) sort the $f_i(1) - f_i(0)$ terms in descending order; and then (2) for each n from 0 to N, compute the sum of the top n entries of the sorted terms plus $f_{\text{card}}(n) + C$ to get the score for n output variables being 1; then (3) choose the maximum score among all $n + 1$ scores computed in (2) and recover the optimal assignment $\mathbf{y}^*$.

A lot of research has gone into developing tractable high order potentials and these models have achieved notable success in computer vision problems [51, 158, 76]. Without exploiting the structure, it is in general intractable to do MAP inference for high order models.

Decomposition methods can be applied when a scoring function is hard to optimize, but it can be decomposed into parts that are easier to optimize individually.

Consider a structured output $\mathbf{y}$, assume two subsets $\mathbf{y}_A$ and $\mathbf{y}_B$ cover all the components in $\mathbf{y}$, A and B are arbitrary subsets of component indices. Suppose the scoring function can be decomposed into two parts

$$F(\mathbf{y}) = F_A(\mathbf{y}_A) + F_B(\mathbf{y}_B), \tag{2.18}$$

and the optimization problem $\operatorname{argmax}_{\mathbf{y}} F(\mathbf{y})$ is hard, but both subproblems $\operatorname{argmax}_{\mathbf{y}_A} F_A(\mathbf{y}_A)$ and $\operatorname{argmax}_{\mathbf{y}_B} F_B(\mathbf{y}_B)$ can be solved efficiently. Further more, let $U = A \setminus B, V = B \setminus A$, and $W = A \cap B$, then $U \cap W = \varnothing$, $V \cap W = \varnothing$, $A = U \cup W$ and $B = V \cup W$.

When the set $\mathbf{y}_W$ is small (not expensive to be enumerated), we can use *primal decomposition* and solve the MAP inference problem as follows,

$$\begin{aligned} \max_{\mathbf{y}} F(\mathbf{y}) &= \max_{\mathbf{y}} \left[F_A(\mathbf{y}_U, \mathbf{y}_W) + F_B(\mathbf{y}_V, \mathbf{y}_W)\right] \\ &= \max_{\mathbf{y}_W} \left[\max_{\mathbf{y}_U} F_A(\mathbf{y}_U, \mathbf{y}_W) + \max_{\mathbf{y}_V} F_B(\mathbf{y}_V, \mathbf{y}_W)\right]. \end{aligned} \tag{2.19}$$

Here the two inner optimization problems can be solved independently and efficiently for a fixed $\mathbf{y}_W$, and the outer optimization can be done by enumeration.

In most cases, enumerating $\mathbf{y}_W$ is infeasible, and we can instead use *dual decomposition*, which converts the original problem into a constrained optimization problem

$$\begin{aligned} \max_{\mathbf{y}_U, \mathbf{y}_V, \mathbf{y}_W, \mathbf{y}'_W} \quad & F_A(\mathbf{y}_U, \mathbf{y}_W) + F_B(\mathbf{y}_V, \mathbf{y}'_W) \\ \text{s.t.} \quad & \mathbf{y}_W = \mathbf{y}'_W \end{aligned} \tag{2.20}$$

The dual problem is

$$\min_{\boldsymbol{\lambda}_W} g(\boldsymbol{\lambda}_W) = \min_{\boldsymbol{\lambda}_W} \left\{ \max_{\mathbf{y}_U, \mathbf{y}_W} \left[F_A(\mathbf{y}_U, \mathbf{y}_W) + \boldsymbol{\lambda}_W^\top \mathbf{y}_W\right] + \max_{\mathbf{y}_V, \mathbf{y}_W} \left[F_B(\mathbf{y}_V, \mathbf{y}_W) - \boldsymbol{\lambda}_W^\top \mathbf{y}_W\right] \right\}. \tag{2.21}$$

Here the $\boldsymbol{\lambda}_W$'s are the Lagrangian multipliers. For each $\boldsymbol{\lambda}_W$, the dual objective $g(\boldsymbol{\lambda}_W)$ provides an upper bound on the primal optimum, solving the dual problem therefore finds the tightest upper bound.

In the dual problem, the two inner subproblems

$$\max_{\mathbf{y}_U, \mathbf{y}_W} \left[F_A(\mathbf{y}_U, \mathbf{y}_W) + \boldsymbol{\lambda}_W^\top \mathbf{y}_W\right] \quad \text{and} \quad \max_{\mathbf{y}_V, \mathbf{y}_W} \left[F_B(\mathbf{y}_V, \mathbf{y}_W) - \boldsymbol{\lambda}_W^\top \mathbf{y}_W\right] \tag{2.22}$$

can both be solved efficiently, as the two $\boldsymbol{\lambda}_W^\top \mathbf{y}_W$ terms are both unary potentials and do not change the structure of the subproblems.

The outer optimization in the dual problem is a continuous and convex optimization problem, and can be solved by, for example, gradient descent. When the optimal solution to two subproblems agree on the value of $\mathbf{y}_W$, the dual objective equals the primal objective which guarantees that the optimal $\mathbf{y}$ for the dual problem is also primal optimal [148]. When the agreement cannot be reached, some heuristics can be used to recover an approximate solution.

These dual decomposition techniques can be further generalized to handle inference on more general models, and be used to develop message passing algorithms [82, 147, 42].

Challenges The types of models with tractable MAP solutions are still quite limited. As mentioned earlier, even for a model with only pairwise potentials, in most cases the MAP inference problem is intractable, and provably hard. On the other hand, approximate solutions may be obtained efficiently. Examples of these approximate solutions include the graph-cut based move-making algorithm for inference in pairwise models, and the beam-search algorithm widely used for language-related tasks.

Overall, MAP inference for structured output models is a very hard problem. The intractable in-

ference in many cases is holding back the potential for these structured output models. Developing expressive structured models therefore requires careful balance between model complexity and inference complexity.

2.3.2 Probabilistic Inference

For probabilistic models, it is also possible to do probabilistic inference, which provides some uncertainty measure in addition to the single MAP solution found by MAP inference.

Marginal inference computes the marginal distribution for subsets of $\mathbf{y}$. For a certain set of components $\mathbf{y}_C$, the marginal distribution is defined as

$$p(\mathbf{y}_C|\mathbf{x}) = \sum_{\mathbf{y}_{-C}} p(\mathbf{y}|\mathbf{x}) = \frac{\sum_{\mathbf{y}_{-C}} F(\mathbf{y})}{\sum_{\mathbf{y}} \exp(F(\mathbf{y}))}, \tag{2.23}$$

where $-C$ is the set of indices of all components not in $\mathbf{y}_C$, and here we again omitted the dependence on $\mathbf{x}$ and $\boldsymbol{\theta}$ from the scoring functions. Such marginal distributions are useful for studying the predictions for subsets of outputs, and are used in learning probabilistic models.

The marginal distributions are in general intractable to compute, as both sums in the denominator and numerator are computed across exponentially many terms. In some special cases, for example in a chain model or tree model, however, the marginal distributions can be computed exactly via a dynamic programming process similar to the one used for MAP inference presented in Section 2.3.1. The whole process can be derived for marginal inference by replacing the max with $\sum$, add with multiplication, and log-domain additive potential functions with multiplicative factors. For some graphical models that are close to a tree but contain loops, it may be possible to use a similar *junction tree algorithm* [95] for inference.

In many cases marginal inference is harder than MAP inference. For example, the submodular pairwise potential model has tractable MAP inference via graph-cuts, but computing the marginal distribution is #P- complete [66].

The most popular marginal inference approximation method is the *variational inference* method [9, 179, 171, 82, 170, 147, 42]. In variational inference, an approximate and structurally simpler distribution q is optimized to approximate the true distribution $p(\mathbf{y}|\mathbf{x})$ as close as possible. The simplest variational inference method is the *mean field* algorithm, which assumes a fully factorial approximate distribution $q(\mathbf{y}) = \prod_i q_i(y_i)$. These approximate q distributions are obtained to minimize the KL-divergence $\mathrm{KL}(q\|p)$

$$KL(q\|p) = \sum_{\mathbf{y}} q(\mathbf{y}) \log \frac{q(\mathbf{y})}{p(\mathbf{y}|\mathbf{x})}. \tag{2.24}$$

For a fully factorial q distribution, and an undirected model defined through scoring functions $p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\exp(\sum_C f_C(\mathbf{y}_C))$, the KL-divergence can be further simplified to

$$KL(q\|p) = \sum_i \sum_{y_i} q(y_i) \log q(y_i) - \sum_C \sum_{\mathbf{y}_C} q(\mathbf{y}_C) f_C(\mathbf{y}_C) + Z, \tag{2.25}$$

where $q(\mathbf{y}_C) = \prod_{i \in C} q_i(y_i)$. Here all the sums are over subsets of $\mathbf{y}$, and are a lot easier to compute than the sum over all possible $\mathbf{y}$'s, as the space of $\mathbf{y}_C$ is a lot smaller than the space of $\mathbf{y}$. This KL-divergence

can be minimized with respect to q through iteratively applying coordinate descent style updates, where in each step one q_i component is updated while all the other $q_j, j \neq i$ are fixed. For the pairwise scoring function in Eq. 2.2, these coordinate descent updates are given by

$$q_i(y_i) \propto \exp\left(f_i(y_i) + \sum_{j:(i,j)\in\mathcal{E}} \sum_{y_j} q_j(y_j) f_{ij}(y_i, y_j)\right). \tag{2.26}$$

Using better approximations that keep more structures, rather than making the fully factorial assumption, it is possible to develop better inference algorithms, like loopy belief propagation [179], and tree-reweighted message passing [171, 79].

In loopy belief propagation, for example, the following messages are computed for each node i and factor ψ associated with clique C:

$$m_{i\to\psi_C}(y_i) = \prod_{C':i\in C', C'\neq C} m_{\psi_{C'}\to i}(y_i) \tag{2.27}$$

$$m_{\psi_C\to i}(y_i) = \sum_{\mathbf{y}_{C\setminus\{i\}}} \psi_C(\mathbf{y}_C) \prod_{i':i'\in C, i'\neq i} m_{i'\to\psi_C}(y'_i). \tag{2.28}$$

The marginal distributions are then computed as

$$q(\mathbf{y}_C) \propto \psi_C(\mathbf{y}_C) \prod_{i, C':i\in C, C'\neq C} m_{\psi_{C'}\to i}(y_i) \tag{2.29}$$

$$q(y_i) \propto \prod_{C:i\in C} m_{\psi_C\to i}(y_i) \tag{2.30}$$

These iterative updates can be derived from generalizations of the dynamic programming algorithm used for chain and tree graphs, but can also be derived in different ways [179, 170, 82, 147, 42].

Sampling is another popular approach for doing probabilistic inference. It uses a set of samples from $p(\mathbf{y}|\mathbf{x})$ to represent the distribution, and do various computations, for example computing expectations, using the samples. A good review of using sampling methods for probabilistic inference can be found in [124].

A lot of sampling methods are based on Markov chain Monte Carlo. Instead of directly generating samples from a distribution, these methods start from an initial sample, which may not be from the desired distribution, and then iteratively and stochastically transitions it through a Markov chain. In general, for a distribution $p(\mathbf{y})$ (which can be the conditioned on $\mathbf{x}$, but here we present the more general case), the transition process from step t to step $t+1$ is modeled through a transition distribution

$$T(\mathbf{y}^{(t+1)}, \mathbf{y}^{(t)}) = p(\mathbf{y}^{(t+1)}|\mathbf{y}^{(t)}) \tag{2.31}$$

The transition distribution T does not depend on t, and such a Markov chain is called to be *homogeneous*. It is shown that, if the transition process leaves a distribution $p^*(\mathbf{y})$ invariant, i.e. transforming p^* using T gets back the same p^*, or mathematically

$$\sum_{\mathbf{y}} T(\mathbf{y}', \mathbf{y}) p^*(\mathbf{y}) = p^*(\mathbf{y}') \tag{2.32}$$

for any $\mathbf{y}'$, then under mild conditions, iteratively applying the transition process to a distribution $p(\mathbf{y})$ will make it converge to the distribution $p^*(\mathbf{y})$ [124]. On a more operational level, starting with an initial sample from an arbitrary distribution, and then iteratively applying the transition process and convert the sample from $\mathbf{y}^{(t)}$ to $\mathbf{y}^{(t+1)}$ according to the transition distribution, after a number of iterations the $\mathbf{y}$ will be a sample from the target distribution $p^*(\mathbf{y})$.

A usually used sufficient condition for $p^*(\mathbf{y})$ being invariant under T is the *detailed balance* property

$$T(\mathbf{y}', \mathbf{y})p^*(\mathbf{y}) = T(\mathbf{y}, \mathbf{y}')p^*(\mathbf{y}'), \quad \forall \mathbf{y}, \mathbf{y}' \tag{2.33}$$

It is easy to see that if p^* satisfies the detailed balance then it will be invariant under T.

The application of Markov chain Monte Carlo methods relies on a good transition distribution T. The Metropolis-Hastings algorithm [117, 53] provides a general mechanism to design transition distributions. In this algorithm, a proposal distribution $q(\mathbf{y}'|\mathbf{y})$ is used to propose moves from $\mathbf{y}$ to a new state $\mathbf{y}'$. Unlike the transition distribution T, the proposal distribution does not have to satisfy detailed balance or have p^* invariant under it, which gives us a lot more flexibility for designing good q's. To guarantee the correctness of the algorithm, the proposal is accepted with probability

$$A(\mathbf{y}', \mathbf{y}) = \min\left\{1, \frac{p^*(\mathbf{y}')q(\mathbf{y}|\mathbf{y}')}{p^*(\mathbf{y})q(\mathbf{y}'|\mathbf{y})}\right\} \tag{2.34}$$

The full transition distribution is therefore

$$T(\mathbf{y}', \mathbf{y}) = \begin{cases} A(\mathbf{y}', \mathbf{y})q(\mathbf{y}'|\mathbf{y}), & \mathbf{y} \neq \mathbf{y}' \\ q(\mathbf{y}|\mathbf{y}) + \sum_{\hat{\mathbf{y}}}(1 - A(\hat{\mathbf{y}}, \mathbf{y}))q(\hat{\mathbf{y}}|\mathbf{y}), & \mathbf{y} = \mathbf{y}' \end{cases}. \tag{2.35}$$

With the acceptance probability defined above, we can show that the transition distribution satisfies the detailed balance. For $\mathbf{y} = \mathbf{y}'$, the detailed balance holds trivially; for $\mathbf{y} \neq \mathbf{y}'$, we have

$$T(\mathbf{y}', \mathbf{y})p^*(\mathbf{y}) = A(\mathbf{y}', \mathbf{y})q(\mathbf{y}'|\mathbf{y})p^*(\mathbf{y}) = \min\left\{p^*(\mathbf{y})q(\mathbf{y}'|\mathbf{y}), p^*(\mathbf{y}')q(\mathbf{y}|\mathbf{y}')\right\} = T(\mathbf{y}, \mathbf{y}')p^*(\mathbf{y}') \tag{2.36}$$

Therefore Metropolis-Hastings algorithm is a valid Markov chain Monte Carlo sampling algorithm.

A special case of the Metropolis-Hastings algorithm is the Gibbs sampling [41] algorithm. In Gibbs sampling, a new $\mathbf{y}'$ is proposed by first choosing an arbitrary index i, and then changing y_i to y_i' with probability $p^*(y_i'|\mathbf{y}_{-i})$, where $\mathbf{y}_{-i}$ is the set of all components in $\mathbf{y}$ except y_i. Therefore after the transition

$$y_j' = y_j, \ \forall j \neq i, \quad \text{and} \quad y_i' \sim p^*(y_i'|\mathbf{y}_{-i}). \tag{2.37}$$

The acceptance probability is then

$$\begin{aligned} A(\mathbf{y}', \mathbf{y}) &= \min\left\{1, \frac{p^*(\mathbf{y}')p^*(y_i|\mathbf{y}'_{-i})}{p^*(\mathbf{y})p^*(y_i'|\mathbf{y}_{-i})}\right\} = \min\left\{1, \frac{p^*(y_i', \mathbf{y}_{-i})p^*(y_i|\mathbf{y}_{-i})}{p^*(\mathbf{y})p^*(y_i'|\mathbf{y}_{-1})}\right\} \\ &= \min\left\{1, \frac{p^*(y_i'|\mathbf{y}_{-i})p^*(\mathbf{y})}{p^*(\mathbf{y})p^*(y_i'|\mathbf{y}_{-i})}\right\} = 1 \end{aligned} \tag{2.38}$$

So using this special transition process, the proposals will always be accepted. In practice when applying Gibbs sampling, we loop over all i's and update y_i to y_i' conditioned on all other $\mathbf{y}_{-i}$ iteratively. This is equivalent to chaining a serious of such transition steps, one for each i, and each of them satisfy detailed

balance, which implies the whole process still maintains p^* as the invariant distribution [9].

In structured models, if conditional independence property $p(y_i, y_j|\mathbf{y}_{-ij}) = p(y_i|\mathbf{y}_{-ij})p(y_j|\mathbf{y}_{-ij})$ holds for y_i and y_j, then the Gibbs steps for updating y_i and y_j can be done in parallel as they do not depend on each other. Using this property it is possible to speed up the sampling process by dividing the set of components in $\mathbf{y}$ into groups, and when updating each group all components inside it is updated together at once conditioned on other groups. This is called the *block Gibbs sampling* method. This grouping speeds up sampling because of the parallelization.

Once a set of samples is obtained, computation on the distribution $p(\mathbf{y}|\mathbf{x})$ can usually be done by computing sample statistics.

Challenges Probabilistic inference is also very challenging. As discussed above, marginal inference is in many cases even harder than MAP inference. Variational inference always makes simplifying assumptions about the approximate distribution, and the quality of the approximations can be far from satisfactory.

Sampling methods are appealing because asymptotically we are guaranteed to get exact samples from the target distribution. However, getting to that stage may take infeasibly many steps. Additionally, sampling methods are stochastic, bad designs of sampling process may lead to results with high variance, which is undesirable.

2.3.3 Separation of Modeling and Inference

Overall, inference for structured output models is a very hard problem. The tradeoff between the expressive power of a model and the complexity of inference should be always considered carefully when designing new models.

The separation of modeling and inference is a key feature of this approach, where modeling is the process that specifies what a desirable prediction looks like, and inference is the process that finds the most desirable prediction. This separation is convenient for developing and adopting new models, but may also be too strict for developing efficient models. The alternative approach that combines modeling and inference, and turns structured models into an inference pipeline, may promise more efficient models with more expressive power.

2.4 Learning

Learning is the process of estimating model parameters $\boldsymbol{\theta}$ based on a training set. In most cases learning $\boldsymbol{\theta}$ is done by optimizing some objective function, which measures how good the model fits the training data. The task loss which is used to evaluate how well the model predictions match the ground truth provides a natural objective, but these task losses are usually complicated and even non-differentiable for structured discrete output objects, making optimization challenging. The Hamming loss $\Delta(\mathbf{y}, \mathbf{y}')$ which counts the number of components in $\mathbf{y}$ different from that of $\mathbf{y}'$ is one such example of non-differentiable task losses. Another example of a more challenging task loss is the Intersection over Union (IoU) score [37] used for evaluating segmentations. Therefore surrogate losses that have nicer properties like differentiability are usually used in place of task losses. When these surrogate losses are reduced, the task losses are usually also reduced as a result, therefore minimizing surrogate losses can provide meaningful training signals for the model.

There are two major frameworks for learning structured output models: Structured Support Vector Machines (SSVMs) [161, 159, 162] and Conditional Random Fields (CRFs) [93]. SSVMs optimize the structured hinge loss, suitable for a purely scoring function based model, and CRFs optimize the negative conditional log-likelihood for a probabilistic model.

In Structured SVMs, the following objective function is minimized

$$\min_{\boldsymbol{\theta}} \quad J(\boldsymbol{\theta}) = \frac{C}{2}\|\boldsymbol{\theta}\|^2 + \frac{1}{N}\sum_n \left\{ \max_{\mathbf{y}} \left[F_{\boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}^n)\right] - F_{\boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}^n) \right\}, \tag{2.39}$$

where $\{(\mathbf{x}^n, \mathbf{y}^n)\}_{n=1}^N$ are example input-output pairs in the training set, $\Delta(\mathbf{y}, \mathbf{y}^*)$ is the task loss which measures how far the prediction $\mathbf{y}$ is from the ground truth $\mathbf{y}^*$, and $\frac{C}{2}\|\boldsymbol{\theta}\|^2$ is a regularizer with weight C.

The loss function

$$\ell(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \max_{\mathbf{y}'} \left[F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}') + \Delta(\mathbf{y}', \mathbf{y})\right] - F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) \tag{2.40}$$

is the *structured hinge loss*. This loss function is minimized when for each pair of ground truth $(\mathbf{x}, \mathbf{y})$, all other $\mathbf{y}' \neq \mathbf{y}$ has a score $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}')$ lower than $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})$ by at least $\Delta(\mathbf{y}', \mathbf{y})$, and hence $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})$ is higher than all other $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}')$ as desired. It can be shown that the structured hinge loss upper bounds the true task loss that we will get when using the model with parameter $\boldsymbol{\theta}$ for prediction. Optimizing the structured hinge loss is thus a proper surrogate loss for the task loss.

Training in SSVMs can be done by subgradient descent, the gradient of the objective with respect to $\boldsymbol{\theta}$ is

$$\frac{\partial J}{\partial \boldsymbol{\theta}} = C\boldsymbol{\theta} + \frac{1}{N}\sum_n \left[\frac{\partial F}{\partial \boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}^{n*}) - \frac{\partial F}{\partial \boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}^n)\right], \tag{2.41}$$

where

$$\mathbf{y}^{n*} = \operatorname*{argmax}_{\mathbf{y}} \left[F(\mathbf{x}^n, \mathbf{y}) + \Delta(\mathbf{y}, \mathbf{y}^n)\right]. \tag{2.42}$$

This optimization problem over $\mathbf{y}$ is very similar to the inference problem where we find $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})$, the only difference being that the task loss Δ is added to F for inference. Therefore this problem is also called the *loss-augmented inference* problem.

For a Hamming loss $\Delta(\mathbf{y}, \mathbf{y}^n)$, the loss itself can be decomposed as a sum $\Delta(\mathbf{y}, \mathbf{y}^n) = \sum_i \mathbf{I}[y_i \neq y_i^n]$, where $\mathbf{I}[.]$ is the indicator function that equals 1 if the condition is true and 0 otherwise. When adding such a loss to the F function, it is equivalent to changing the unary potentials, therefore the loss-augmented inference can be solved with the same inference algorithm as used for the standard inference problem.

In general, the loss-augmented inference problem is only solvable when the loss $\Delta(\mathbf{y}, \mathbf{y}')$ has some special structures, like decompositionality. In such cases, Δ is equivalent to an additional potential function, or a sum of potential functions, and the same class of inference algorithms can be used to solve the loss-augmented inference problem. In some cases, by exploiting the structure of the losses, high order loss functions that are not decomposable may also be optimized [157].

Conditional Random Fields optimizes the negative log-likelihood objective

$$\min_{\boldsymbol{\theta}} \quad J(\boldsymbol{\theta}) = \frac{C}{2}\|\boldsymbol{\theta}\|^2 - \frac{1}{N}\sum_n \log p_{\boldsymbol{\theta}}(\mathbf{y}^n|\mathbf{x}^n), \tag{2.43}$$

where the distribution $p_{\boldsymbol{\theta}}$ is defined using $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})$ as

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \frac{\exp(F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}))}{\sum_{\mathbf{y}'} \exp(F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}'))}. \tag{2.44}$$

Again regularizers can be properly added to the objective function.

The gradients of this objective function can be computed as

$$\frac{\partial J}{\partial \boldsymbol{\theta}} = C\boldsymbol{\theta} + \frac{1}{N} \sum_n \left\{ \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^n)} \left[\frac{\partial F}{\partial \boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}) \right] - \frac{\partial F}{\partial \boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}^n) \right\}, \tag{2.45}$$

where $\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^n)}[.]$ is an expectation under the conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^n)$ given by the current model. Once this gradient is computed, we can use various optimization techniques, e.g. gradient descent, to learn the optimal $\boldsymbol{\theta}$.

The challenge of this approach lies in estimating the expectation term. Since

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^n)} \left[\frac{\partial F}{\partial \boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}) \right] = \sum_{\mathbf{y}} p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^n) \frac{\partial F}{\partial \boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}), \tag{2.46}$$

computing the expectation by summing over $\mathbf{y}$ is hard, as $\mathbf{y}$ usually lives in an exponentially large space. For models with structure however, this expectation may be simplified. For example for the pairwise scoring function given in Eq. 2.2, the expectation decomposes into expectations under marginal distributions over individual y_i's and pairs of (y_i, y_j)'s

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^n)} \left[\frac{\partial F}{\partial \boldsymbol{\theta}}(\mathbf{x}^n, \mathbf{y}) \right] = \sum_i \mathbb{E}_{p_{\boldsymbol{\theta}}(y_i|\mathbf{x}^n)} \left[\frac{\partial f_i}{\partial \boldsymbol{\theta}}(y_i, \mathbf{x}^n) \right] + \sum_{i,j} \mathbb{E}_{p_{\boldsymbol{\theta}}(y_i, y_j|\mathbf{x}^n)} \left[\frac{\partial f_{ij}}{\partial \boldsymbol{\theta}}(y_i, y_j, \mathbf{x}^n) \right]. \tag{2.47}$$

Here the expectations over marginal distributions are over either a single variable or a pair of variables, therefore computing the sum becomes easier. The probabilistic inference methods proposed in the previous section can be used to compute these expectations. Variational inference approximates the original distribution with factorized distributions, and compute approximate marginals. These approximate marginals can be used to compute the expectations. Sampling methods, on the other hand, generates samples from the model distribution, and computes the expectations by averaging across a set of samples.

Task loss can be incorporated in CRFs to make the training adaptive to the actual loss. This can be done by changing the conditional distribution in CRFs to

$$\hat{p}_{\boldsymbol{\theta}}(\hat{\mathbf{y}}|\mathbf{x}, \mathbf{y}) = \frac{\exp(F_{\boldsymbol{\theta}}(\mathbf{x}, \hat{\mathbf{y}}) + \Delta(\hat{\mathbf{y}}, \mathbf{y}))}{\sum_{\mathbf{y}'} \exp(F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}') + \Delta(\mathbf{y}', \mathbf{y}))}. \tag{2.48}$$

Further, in [54], it was shown that CRFs with such loss-augmented formulation can be unified with the SSVMs in a single framework, with a temperature parameter ϵ that controls the smoothness of the loss. The CRF formulation is obtained as a special case of $\epsilon = 1$ and the SSVM formulation corresponds to $\epsilon = 0$.

For the special case of log-linear models, both the SSVM and CRF gradients can be simplified a little

bit:

$$\text{SSVM}: \quad \frac{\partial J}{\partial \boldsymbol{\theta}} = C\boldsymbol{\theta} + \frac{1}{N}\sum_n [\boldsymbol{\phi}(\mathbf{x}^n, \mathbf{y}^{n*}) - \boldsymbol{\phi}(\mathbf{x}^n, \mathbf{y}^n)] \tag{2.49}$$

$$\text{CRF}: \quad \frac{\partial J}{\partial \boldsymbol{\theta}} = C\boldsymbol{\theta} + \frac{1}{N}\sum_n \left\{\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^n)}[\boldsymbol{\phi}(\mathbf{x}^n, \mathbf{y})] - \boldsymbol{\phi}(\mathbf{x}^n, \mathbf{y}^n)\right\} \tag{2.50}$$

Furthermore, the SSVM objective Eq. 2.39 and the CRF objective Eq. 2.43 are both convex in $\boldsymbol{\theta}$ for log-linear models. Global minimum of the training objectives can therefore be found with (sub)gradient descent and other proper optimization methods.

Overall, learning in structured output models is an even harder task than inference, as inference, either the discrete optimization in loss-augmented inference for SSVMs or the probabilistic inference in CRFs, is usually called as a subroutine in learning. The approximations made in the inference algorithms may negatively affect the learning process. Thinking about the inference and learning problems together can lead to overall better models [169, 89, 150, 32, 33].

2.5 Neural Network Models

Neural networks are a general class of models for computing a mapping from inputs to outputs. Mathematically speaking, a neural network model that maps an input $\mathbf{x} \in \mathcal{X}$ to an output $\mathbf{y} \in \mathcal{Y}$ can be formulated as

$$\mathbf{y} = F_{\boldsymbol{\theta}}(\mathbf{x}), \tag{2.51}$$

where F is a usually nonlinear function of $\mathbf{x}$, parameterized by $\boldsymbol{\theta}$. Common neural networks consist of *layers* of computations. The results of the computation at layer l, $\mathbf{h}^l \in \mathbb{R}^{H_l}$, is computed from the previous layer $\mathbf{h}^{l-1}$ as

$$\mathbf{h}^l = f_l(\mathbf{h}^{l-1}, \boldsymbol{\theta}^l). \tag{2.52}$$

$\boldsymbol{\theta}^l$ is the parameters associated with the lth layer, and f_l is a transformation function. Each dimension of $\mathbf{h}^l$ is sometimes called a *unit* on layer l. Typical transformation functions have the form of

$$f_l(\mathbf{h}^l, \boldsymbol{\theta}^l) = \sigma\left(\mathbf{W}^l\mathbf{h}^l + \mathbf{b}^l\right) \tag{2.53}$$

where $\mathbf{W}^l \in \mathbb{R}^{H_l \times H_{l-1}}$ and $\mathbf{b}^l \in \mathbb{R}^{H_l}$ are parameters, i.e. $\boldsymbol{\theta}^l = \text{vec}([\mathbf{W}^l, \mathbf{b}^l])$ where vec(.) vectorizes the parameters, and σ is an element-wise and usually nonlinear *activation* function. Typical choices for σ include the logistic sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, the tanh function $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and the rectified linear (ReLU) function $\sigma(x) = \max\{0, x\}$. As a convention, we have $\mathbf{h}^0 = \mathbf{x}$ and $\mathbf{h}^L = \mathbf{y}$ with $\mathbf{x}$ and $\mathbf{y}$ properly vectorized for a network with L layers, $F_{\boldsymbol{\theta}}$ is therefore defined as

$$F_{\boldsymbol{\theta}}(\mathbf{x}) = f_L(f_{L-1}(\cdots f_1(\mathbf{x}, \boldsymbol{\theta}^1)\cdots, \boldsymbol{\theta}^{L-1}), \boldsymbol{\theta}^L) \tag{2.54}$$

Networks with this formulation is called *feed-forward* and *fully-connected* networks. The term feed-forward refers to not having recurrent connections which loop back to lower layers from higher layers in the model, and the term fully-connected means that all components in one layer is connected to all components in the next. Adding recurrent connections and using sparse connection structures are ways to adapt neural networks to structured domains.

Neural networks can be trained with gradient-based optimization methods to optimize a particular objective function. The gradient of the objective function with respect to the network parameters can be obtained via the chain rule using the backpropagation algorithm [139].

Neural networks can be used in various ways to solve structured problems. In the standard structured output learning model, neural networks can be used to model the potential functions as mentioned in Section 2.2. In this section, however, we discuss the neural network models as an *alternative* to the standard structured output learning models.

In the formulation of standard structured output models outlined in Section 2.1, the structured model is defined via the scoring function $F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})$, while the prediction is done by an inference algorithm that optimizes the scoring function with respect to $\mathbf{y}$, i.e. $\mathbf{y}^* = \operatorname{argmax}_{\mathbf{y}}, F_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})$, or runs probabilistic inference on $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})$, which uses $F_{\boldsymbol{\theta}}$ in an indirect way. Here, neural networks can instead model the prediction process directly through the network function $\mathbf{y} = F_{\boldsymbol{\theta}}(\mathbf{x})$. When predicting discrete output objects, the output $\mathbf{y}$ is usually processed via a simple procedure, like taking argmax for each group of values in $\mathbf{y}$, rather than computed from a more complicated optimization process in standard structured output learning.

There are a number of potential benefits of directly modeling the prediction function $\mathbf{y} = F_{\boldsymbol{\theta}}(\mathbf{x})$: (1) inference is easy, as making a prediction only requires a single function evaluation, which is by construction easy to do; (2) learning does not require an intractable inference procedure as a subroutine, therefore we are guaranteed to be able to compute the correct gradient efficiently; (3) more room for designing more expressive models, as we are not limited to using models with certain constraints because inference is easy.

The main challenge for using neural networks for structured problems is that, such problems involve structured input data and structured output objects, but standard neural networks can not readily deal with them as they require input and outputs to live in $\mathbb{R}^n$. Developing *structured* neural network models that can properly represent and exploit the structures in data is an answer to this challenge.

Recurrent neural networks (RNNs) [174, 63] are a family of models widely used for structured problems, mostly sequence prediction problems. The simplest standard RNN takes a sequence input $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_T\} \in \mathcal{X} = \mathbb{R}^D \times \cdots \times \mathbb{R}^D$ composed of data in T time steps, and produces an output sequence $\mathbf{y} = \{y_1, ..., y_T\} \in \mathcal{Y} = \{1, ..., K\}^T$. Each time step has a hidden state $\mathbf{h}_t \in \mathbb{R}^H$, computed as

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}, \boldsymbol{\theta}_f). \tag{2.55}$$

This computation features connections from $\mathbf{h}_{t-1}$ to $\mathbf{h}_t$, and for each time step this computation is the same with different input $\mathbf{x}_t$, but the same set of parameters and the same operations. Such connections are called *recurrent* connections. The output at time t is computed as a function of $\mathbf{h}_t$

$$y_t = g(\mathbf{h}_t, \boldsymbol{\theta}_g), \tag{2.56}$$

where $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$ are model parameters. The common choice of f usually has the form of

$$\mathbf{h}_t = \sigma(\mathbf{W}^x \mathbf{x}_t + \mathbf{W}^h \mathbf{h}_{t-1} + \mathbf{b}^h), \tag{2.57}$$

where $\mathbf{W}^x \in \mathbb{R}^{H\times D}, \mathbf{W}^h \in \mathbb{R}^{H\times H}$ and $\mathbf{b}^h \in \mathbb{R}^H$ are parameters, and σ is an activation function, while the function g can be implemented with any feed-forward neural network.

RNNs can adapt to sequences of different lengths. The model is usually trained by optimizing a loss of the form $\ell(\mathbf{y}, \mathbf{y}^*) = \sum_t \ell(y_t, y_t^*)$, where each $\ell(y_t, y_t^*)$ is differentiable with respect to model predictions y_t. The parameter gradients can then be computed using the chain-rule, with some extra care to handle the weight sharing across time steps [174].

RNNs have been successfully applied to a range of sequence prediction problems, including speech recognition [48, 47], language modeling [119, 8], machine translation [153, 5], to name just a few. A few variants of RNNs also exist that enhances its ability to handle longer sequences [63, 26, 49, 68].

Convolutional neural networks (CNNs) [39, 97] are another family of structured neural network models popular for visual data. The spatial locality is one of the key features of visual data, and convolutional neural networks model this locality with local connection structure. In a CNN, the input image is a 2D array of pixels, and each layer is also organized as 2D arrays, keeping the spatial locality. Furthermore, each unit in $\mathbf{h}^l$ is computed from only a small local window of $\mathbf{h}^{l-1}$, and the connection weights in this local window are shared across all locations in the layer. The transformation function of a layer in convolutional neural networks can be formulated as

$$\mathbf{h}^l = \sigma(\mathbf{W}^l * \mathbf{h}^{l-1} + \mathbf{b}^l), \tag{2.58}$$

where $*$ is the convolution operator, and parameters $\mathbf{W}^l$ and $\mathbf{b}^l$ have shapes properly suited to the input and output.

Unlike the pairwise models in standard structured output learning, where local information are typically aggregated in a limited way only during the inference procedure, CNNs keep locality in local connections but also aggregate information through layers of computations that processes and exchanges information in a more complicated and learnable computation process. CNNs have achieved great success in a number of computer vision problems, whether for unstructured problems like object classification where a single output is produced [87, 144, 154, 56], or more structured output problems like semantic segmentation [109, 24], where the output is a grid of pixel labelings same size as the input.

Chapter 3

Compositional High Order Pattern Potentials

In standard structured output learning problems, a key research issue is how to best trade off expressivity of the model with the ability to efficiently learn and perform inference (make predictions). Traditionally, these concerns have led to the use of overly simplistic models over labelings that make unrealistic conditional independence assumptions, such as pairwise models with grid-structured topology. In the past few years, there have been successful efforts that weaken these assumptions, either by moving to densely connected pairwise models [84] or by enforcing smoothness in higher order neighborhoods [76]. However, while these approaches can lead to improved performance, they do not capture much higher level structure in the data. Examples of such structures include the shape of an object in computer vision, and high-order sentence structures in natural language understanding. As we look to build models that more faithfully represent structure present in the world, it is desirable to explore the use of models capable of representing this higher level structure. In this chapter, we focus on the computer vision application of image segmentation, but the techniques developed here can also be applied to other applications.

One promising direction towards incorporating these goals in the structured output setting appears to be the *pattern potentials* of Rother et al. [138] and Komodakis and Paragios [81], which are capable of modeling soft template structures and can dramatically outperform pairwise models in highly structured settings that arise, for example, when modeling regular textures. Yet despite the clearly powerful representational ability of pattern potentials, they have not found much success in more realistic settings, like those found in the PASCAL VOC image labeling task [37].

We hypothesize that this is due to three main factors: (a) learning pattern potentials that perform well is difficult; (b) the patterns are not image- or context-specific (i.e., they do not adapt based on local image information); and (c) the primary formulation from [138] assumes only one pattern is active for each patch, which makes it difficult to represent more complex shapes, such as those that are compositional in nature (i.e., they might require exponentially many patterns).

A model that is appropriate in similar situations and has also found success modeling textures [75] is the Restricted Boltzmann Machine (RBM). In fact, our starting observation in this chapter is that the similarity is not superficial—mathematically, RBM models are nearly identical to the pattern potentials of [138], up to choices described in [138] about summing over or minimizing over patterns that

equate to questions of sparsity in the RBM hidden variable activations. We will make this claim precise in Section 3.2, leading to the definition of a more general class of high order potential that includes both pattern potentials and RBMs. We call this class *Compositional High Order Pattern Potentials* (CHOPPs). A primary benefit of this observation is that there is a well-developed literature on learning RBM models that becomes available for learning pattern-like potentials.

In this chapter we explore augmenting standard CRF models with CHOPPs. Our goal is to not only learn a tradeoff parameter between the standard and high order parts of the model, but also to learn internal pattern parameters jointly with other parts of the model. We then focus on the question of how effective these potentials are as the variability and complexity of the image segmentation task increases. We propose a simple method for assessing the degree of variation in the labels, then show that the performance of a vanilla application of CHOPPs degrades relative to the performance of standard pairwise potentials as this measure of variability increases.

We then turn attention to improving vanilla CHOPP-augmented CRFs, and make two primary suggestions. The first is to incorporate additional parameters in the RBM-based potential that allows the pattern activities to depend on information in the image. This is analogous to allowing standard pairwise potentials to vary depending on local image color differences [14] or more advanced boundary detectors like Pb [113]. The second is to utilize a loss function during training that is tailored to the metric used for evaluating the labeling results at test time. Our results indicate that jointly training the CHOPP potentials with the rest of the model improves performance, and training specifically for the evaluation criterion used at test time (we use an intersection-over-union measure throughout) improves over a maximum likelihood-based objective. Finally, we explore (a) different forms of compositionality: the 'min' version advocated by Rother et al. [138], which is essentially a mixture model, versus the 'sum' version, which is more compositional in nature; and (b) convolutional applications of the high order potentials versus their global application.

Since this work sits at the interface of structured output learning and RBM learning, we conclude by suggesting take-aways for both the RBM-oriented researcher and the structured output-oriented researcher, proposing what each approach has to offer the other and outlining possible directions for improving the applicability of pattern-based approaches to challenging structured output problems.

3.1 Related Work

3.1.1 Structured Output Learning

In this section, we review a few more topics on structured output learning that are particularly related to the CHOPPs model: structured output models with latent variables, high order potentials and learning high order potentials. The discussion will be based on the CRF framework, which we build on.

Latent Variable Models To increase the representational power of a model, a common approach is to introduce latent (or hidden) variables $\mathbf{h} \in \mathcal{H} = \{0, \ldots, H-1\}^J$. The standard structured output model formulation presented in Section 2.1 can be easily extended by defining feature functions $f(\mathbf{y}, \mathbf{h}, \mathbf{x})$ that may include latent variables, which leads to a probability distribution $p(\mathbf{y}, \mathbf{h} \,|\, \mathbf{x})$:

$$p(\mathbf{y}, \mathbf{h}|\mathbf{x}) = \frac{\exp(f(\mathbf{y}, \mathbf{h}, \mathbf{x}))}{\sum_{\mathbf{y}', \mathbf{h}'} \exp(f(\mathbf{y}', \mathbf{h}', \mathbf{x}))}. \tag{3.1}$$

To make predictions in latent variable models, it is common to either maximize out or sum out the latent variables:

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}} \max_{\mathbf{h}\in\mathcal{H}} p(\mathbf{y}, \mathbf{h} \mid \mathbf{x}; \boldsymbol{\lambda}), \quad \text{or} \quad \mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}} \sum_{\mathbf{h}\in\mathcal{H}} p(\mathbf{y}, \mathbf{h} \mid \mathbf{x}; \boldsymbol{\lambda}). \tag{3.2}$$

The former strategy is employed by latent structural SVMs [180], while the latter is employed by hidden CRF models [135]. A topic of ongoing investigation is the benefits of each, and alternative strategies that interpolate between the two [120].

High Order Potentials A related strategy for increasing the representational power of a model is to allow feature functions to depend on a large number of dimensions of $\mathbf{y}$. These types of interactions are known collectively as *high order potentials* and have received considerable attention in the last few years. They have been used for several purposes, including modeling higher order smoothness [76], co-occurrences of labels in semantic image segmentation [92], and cardinality-based potentials [166, 177]. While the above examples provide interesting non-local constraints, they do not encode shape-based information appropriate for image labeling applications. There are other high order models that come closer to this goal: modeling star convexity [50], connectivity [165, 127], and a bounding box occupancy constraint [100]. However, these still are quite restrictive notions of shape compared to what pattern-based models are capable of representing. Some more discussion about these high order potentials are presented in Chapter 2.

Learning High Order Potentials In addition to a weighting coefficient that governs the relative contribution of each feature function to the overall scoring function, the features also have internal parameters. This is the case in CHOPPs, where internal parameters dictate the target pattern and the costs for deviating from it. These parameters also need to be set, and the approach we take in this chapter is to learn them. We emphasize the distinction between first learning the internal parameters offline and then learning (or fixing by hand) the trade-off parameters that controls the relative strength of the high order terms, versus the joint learning of both types of parameters. While there is much work that takes the former approach [80, 138, 92, 91], there is little work on the latter in the context of high order potentials. Indeed it is more challenging, as standard learning formulations become less appropriate , and objectives are generally non-convex.

3.1.2 Pattern Potentials

Pattern potentials are proposed by Rother et al. [138] as an expressive high order potential used for image labeling problems. Unlike most other high order potentials proposed in the past, pattern potentials take a data driven approach, which specify preferences for the image labels to be close to a set of *patterns* learned from data. The patterns used in pattern potentials are quite general, and can take arbitrary forms. Therefore these pattern potential models are very expressive and can be applied to a range of different applications. In image segmentation problems in particular, pattern potentials are useful to represent the shape of objects, as did in our work and a few others [69, 178]. In [138], pattern potentials are also demonstrated to perform very well in low level vision tasks like texture restoration from noisy images.

As mentioned in the previous subsection, the patterns in pattern potentials are learned offline by a clustering algorithm, and then fixed and used jointly with other parts of a CRF model for inference and learning of a linear weighting model. In our work, we instead learn these patterns jointly with other parts of the CRF, and therefore can achieve further improvement over offline-learned patterns.

A more detailed discussion about the formulation of pattern potentials can be found in Section 3.2.

3.1.3 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) [145, 38, 61] is a form of undirected graphical model that uses hidden variables to model high-order regularities in data. It consists of the I *visible* units $\mathbf{v} = (v_1, \ldots, v_I)^\top$ that represent the observations, or data; and the J *hidden* or latent units $\mathbf{h} = (h_1, \ldots, h_J)^\top$ that mediate dependencies between the visible units. The system can be seen as a bipartite graph, with the visibles and the hiddens forming two layers of vertices in the graph; the restriction is that no connection exists between units in the same layer.

The aim of the RBM is to represent probability distributions over the states of the random variables. The pattern of interaction is specified through the energy function:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^\top \mathbf{W}\mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h} \tag{3.3}$$

where $\mathbf{W} \in \mathbb{R}^{I \times J}$ encodes the hidden-visible interactions, $\mathbf{b} = (b_1, \ldots, b_I)^\top$ the input and $\mathbf{c} \in \mathbb{R}^J$ the hidden self-connections (also known as biases). The energy function specifies the probability distribution over the joint space $(\mathbf{v}, \mathbf{h})$ via the Boltzmann distribution

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{3.4}$$

with the partition function Z given by $\sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$. In our formulation using scoring functions, the scoring function for RBMs is then

$$f(\mathbf{v}, \mathbf{h}) = -E(\mathbf{v}, \mathbf{h}) = \mathbf{v}^\top \mathbf{W}\mathbf{h} + \mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h}. \tag{3.5}$$

Based on this definition, the probability for any subset of variables can be obtained by conditioning and marginalization. In particular, due to the bipartite graph structure, both of the conditional distributions $p(\mathbf{h}|\mathbf{v})$ and $p(\mathbf{v}|\mathbf{h})$ are factorial, and have the following form:

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) = \sigma\left(c_j + \sum_i w_{ij} v_i\right) \quad \text{and} \quad p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) = \sigma\left(b_i + \sum_j w_{ij} h_j\right) \tag{3.6}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic sigmoid function.

Learning in RBMs For maximum likelihood learning, the goal is to make the data samples likely, which entails computing the probability for any input $\mathbf{v}$; this can be derived by performing the exponential sum over all possible hidden vectors $\mathbf{h}$: $p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})$, effectively marginalizing them out. For an RBM with I binary visible units, this takes on a particular nice form:

$$\begin{aligned} p(\mathbf{v}) &= \sum_{\mathbf{h}} \frac{1}{Z} \exp(\mathbf{v}^\top \mathbf{W}\mathbf{h} + \mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h}) \\ &= \frac{1}{Z} \exp\left(\mathbf{b}^\top \mathbf{v} + \sum_j \log\left(1 + \exp(\mathbf{v}^\top \mathbf{w}_j + c_j)\right)\right) \end{aligned} \tag{3.7}$$

where $\mathbf{w}_j$ is the jth column in $\mathbf{W}$ and each of the terms inside the summation over j is known as a

softplus, and each terms effectively forms a high order potential over $\mathbf{v}$. Since it is hard to compute Z, exact maximum likelihood learning is intractable. The standard approach to learning in RBMs uses an approximation to maximum likelihood learning known as Contrastive Divergence (CD) [61]. More specifically, the marginal log-likelihood can be expanded as

$$\ell = \log p(\mathbf{v}) = \log \sum_{\mathbf{h}} \exp(\mathbf{v}^\top \mathbf{W}\mathbf{h} + \mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h}) - \log \sum_{\mathbf{v},\mathbf{h}} \exp(\mathbf{v}^\top \mathbf{W}\mathbf{h} + \mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h}) \tag{3.8}$$

The gradient with respect to parameter vector $\mathbf{W}$ is

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{W}} &= \sum_{\mathbf{h}} \frac{\exp(\mathbf{v}^\top \mathbf{W}\mathbf{h} + \mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h})}{\sum_{\mathbf{h}'} \exp(\mathbf{v}^\top \mathbf{W}\mathbf{h}' + \mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h})} \mathbf{v}\mathbf{h}^\top - \sum_{\mathbf{v},\mathbf{h}} \frac{\exp(\mathbf{v}^\top \mathbf{W}\mathbf{h} + \mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h})}{\sum_{\mathbf{v}',\mathbf{h}'} \exp(\mathbf{v}'^\top \mathbf{W}\mathbf{h}' + \mathbf{b}^\top \mathbf{v}' + \mathbf{c}^\top \mathbf{h}')} \mathbf{v}\mathbf{h}^\top \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \mathbf{v}\mathbf{h}^\top - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) \mathbf{v}\mathbf{h}^\top \\
&= \mathbb{E}_{p(\mathbf{h}|\mathbf{v})}[\mathbf{v}\mathbf{h}^\top] - \mathbb{E}_{p(\mathbf{v},\mathbf{h})}[\mathbf{v}\mathbf{h}^\top].
\end{aligned} \tag{3.9}$$

The first expectation is computed for a given data vector $\mathbf{v}$ and the expectation is taken over the hidden states $\mathbf{h}$. Since the posterior distribution $p(\mathbf{h}|\mathbf{v})$ factorizes as in Eq. 3.6, the first expectation is easy to compute. The second expectation is taken with respect to the joint model distribution over both $\mathbf{v}$ and $\mathbf{h}$, and is hard to compute.

In CD learning, the second expectation is estimated using samples from the model. Samples are generated with block Gibbs sampling, a Markov chain is usually started at the data vector $\mathbf{v}$, then hidden states are sampled according to $p(\mathbf{h}|\mathbf{v})$, and then data vectors $\mathbf{v}$ are updated again according to $p(\mathbf{v}|\mathbf{h})$ which is then done iteratively for up to T rounds. When the sampling Markov chain is run for T rounds, the CD variant is called CD-T. After the samples are obtained, the expectation over $p(\mathbf{v},\mathbf{h})$ is computed using sample average.

Vision Applications There has been numerous applications of RBM to vision problems. RBMs are typically trained to model the input data such as an image, and most vision applications have focused on this standard unsupervised training paradigm. For example, they have been used to model object shape [36], images under occlusion [96], and noisy images [156]. They have also been applied in a discriminative setting, as joint models of inputs and a class [94].

The focus of the RBMs we explore here, as models of image labels, has received relatively little attention. Note that in this case the visible units of the RBM now correspond to the image labels $\mathbf{y}$. The closest work to ours is that of [58]. The RBMs in that work only captured very local or global patterns in the label field, and did not address shape information as we do, and it also combined the RBM with a very restricted form of CRF, which greatly simplified inference and learning. [123] also tried to use RBMs for structured output problems, but the model used in the paper did not have pairwise connections in the label field, and the actual loss was not considered during training. Also related is the work of [35], which uses a generative framework to model labels and images.

3.2 Equating Pattern Potentials and RBMs

This section provides the detailed proof of the equivalence between pattern potentials and RBMs. The high level idea of the proof is to treat each hidden variable in an RBM as encoding a pattern.

Composition Scheme for Pattern Potentials	Operation on RBM		Constraint on $\mathbf{h}$
	Maximizing out $\mathbf{h}$	Summing out $\mathbf{h}$	
Max	$\max_{1\leq j\leq J}\left\{c_j+\sum_{i=1}^{I} w_{ij}y_i\right\}$	$\log\left(1+\sum_{j=1}^{J-1}\exp\left(c_j+\sum_{i\in a} w_{ij}y_i\right)\right)$	1-of-J
Sum	$\sum_{j=1}^{J}\max\left\{c_j+\sum_{i=1}^{I} w_{ij}y_i, 0\right\}$	$\sum_{j=1}^{J}\log\left(1+\exp\left(c_j+\sum_{i=1}^{I} w_{ij}y_i\right)\right)$	None

Table 3.1: Equivalent compositional high order potentials by applying different operations and constraints on RBMs. Maximizing out hidden variables results in high order potentials that are exactly equivalent to pattern potentials. Summing out hidden variables results in approximations to pattern potentials. 1-of-J constraint on hidden variables corresponds to the "max" compositional scheme. No constraints on hidden variables corresponds to "sum" compositional scheme.

We first introduce (in Section 3.2.1) the definition of pattern potentials by Rother et al. [138], a few necessary rearrangements of the definition, some change of variable tricks, and two different ways to compose more general high order potentials, "sum" and "max".

After the necessary preparation, we relate the compositional pattern potentials to RBMs. We show in Section 3.2.2 that the RBM potentials with hidden variables *maximized out* is equivalent to pattern potentials. When there are no constraints on hidden variables, we recover the "sum" compositional pattern potentials; when there is a 1-of-J constraint on hidden variables, we recover the "max" compositional pattern potentials. In Section 3.2.3, we show that *summing out* hidden variables in RBMs approximates pattern potentials, and similarly with and without constraints on hidden variables would lead us to "max" and "sum" cases respectively.

The RBM formulation offers considerable generality via choices about how to constrain hidden unit activations. This allows a smooth interpolation between the "sum" and "max" composition strategies. Also, this formulation allows the application of learning procedures that are appropriate for cases other than just the "max" composition strategy.

In Section 3.3, we provide a way to unify maximizing out and summing out hidden variables by introducing a temperature parameter in the model.

Notation In this section, we use f for pattern potentials and g for the high order potentials induced by RBMs. Superscripts 's' and 'm' on f corresponds to two composition schemes, sum and max. Superscripts on g correspond to two types of constraints on RBM hidden variables, and subscripts on g correspond to maximizing out or summing out hidden variables.

Our results in this section are summarized in Table 3.1.

3.2.1 Pattern potentials

In [138], a basis pattern potential for binary variables $\mathbf{y}\in\{0,1\}^n$ is defined as

$$f(\mathbf{y}) = \max\{\theta_0 - d(\mathbf{y}), 0\} \tag{3.10}$$

where $d:\{0,1\}^n\to[0,+\infty)$ is a deviation function specifying the distance of $\mathbf{y}$ from a specific pattern $\mathbf{y}^p$, and parameter $\theta_0>0$. The pattern potential therefore favors $\mathbf{y}$ to be close to a certain pattern $\mathbf{y}^p$, and the score is lower bounded by 0, i.e. the distance $d(\mathbf{y})$ does not matter any more when $\mathbf{y}$ is too far ($d(\mathbf{y})>\theta_0$) from $\mathbf{y}^p$, while θ_0 is the maximum achievable score. Note that this definition is adapted from [138] to fit in our presentation. In [138] the pattern potentials are defined as penalty functions to

be minimized, while here we use an equivalent definition of them as score functions to be maximized[1].

For a specific pattern (also binary) $\mathbf{y}^p$, the deviation function $d(\mathbf{y})$ is defined as[2]

$$d(\mathbf{y}) = \sum_{i \in a} abs(w_i) \mathbf{I}[y_i \neq y_i^p] \tag{3.11}$$

where $abs()$ is the absolute value function. The parameter w_i specifies the preference for y_i to be assigned to 1, $w_i > 0$ when $y_i^p = 1$ and $w_i < 0$ when $y_i^p = 0$. This is a weighted Hamming distance of $\mathbf{y}$ from $\mathbf{y}^p$. Since $\mathbf{y}$ and $\mathbf{y}^p$ are both binary vectors, we have the following alternative formulation of $d(\mathbf{y})$,

$$\begin{aligned} d(\mathbf{y}) &= \sum_{i:y_i^p=1} w_i(1-y_i) + \sum_{i:y_i^p=0} (-w_i) y_i \\ &= -\sum_i w_i y_i + \sum_{i:y_i^p=1} w_i. \end{aligned} \tag{3.12}$$

Substitute this back into Eq. 3.10, we get

$$f(\mathbf{y}) = \max \left\{ \theta_0 + \sum_i w_i y_i - \sum_{i:y_i^p=1} w_i, 0 \right\} \tag{3.13}$$

Reparameterize θ_0 by using $c = \theta_0 - \sum_{i:y_i^p=1} w_i$, we can rewrite the above equation as

$$f(\mathbf{y}) = \max \left\{ c + \sum_i w_i y_i, 0 \right\} \tag{3.14}$$

This formulation is useful for establishing connections with RBMs as shown later in this section.

[138] proposed two ways to compose more general high order potentials from basis pattern potentials defined above. One is to take the sum of different pattern potentials

$$f^s(\mathbf{y}) = \sum_{j=1}^{J} \max\{\theta_j - d_j(\mathbf{y}), 0\}, \tag{3.15}$$

and the other is to take the maximum ("max") of them, to get

$$f^m(\mathbf{y}) = \max_{1 \leq j \leq J} \{\theta_j - d_j(\mathbf{y})\} \tag{3.16}$$

In both cases, $d_j(.)$'s are J different deviation functions associated with J different patterns. In the "max" case, we can also fix one deviation function to be 0 (e.g. by setting all weights $w_i = 0$), to get a constant threshold.

Using the parameterization with w and c introduced above, we can rewrite the "sum" compositional pattern potential as

$$f^s(\mathbf{y}) = \sum_{j=1}^{J} \max \left\{ c_j + \sum_i w_{ij} y_i, 0 \right\}, \tag{3.17}$$

[1] Another minor difference is that in [138], the lower bound is specified as a parameter $\theta_{\min}$ instead of 0, while here we subtracted it from $f(\mathbf{y})$ as it does not depend on $\mathbf{y}$ and doing this makes the presentation cleaner.

[2] Note that in [138], there is also a factor θ in this definition ($d(\mathbf{y})$ is given by the product of factor θ and the sum), but actually the θ factor can always be absorbed in w_i's to get this equivalent formulation.

and rewrite the "max" compositional pattern potential as

$$f^m(\mathbf{y}) = \max_{1 \le j \le J} \left\{ c_j + \sum_i w_{ij} y_i \right\} \tag{3.18}$$

3.2.2 Maximizing out hidden variables in RBMs

We start from maximizing hidden variables out. The probability distribution defined by a binary RBM is given by

$$p(\mathbf{y}, \mathbf{h}) = \frac{1}{Z} \exp\left(-E(\mathbf{y}, \mathbf{h})\right) \tag{3.19}$$

where the energy

$$E(\mathbf{y}, \mathbf{h}) = -\sum_{i=1}^{I} \sum_{j=1}^{J} w_{ij} y_i h_j - \sum_{i=1}^{I} b_i y_i - \sum_{j=1}^{J} c_j h_j \tag{3.20}$$

Maxing out the hidden variables in the *negative* energy function (Eq. 3.5), the equivalent high order potential is

$$g_{\max}(\mathbf{y}) = \max_{\mathbf{h}} \left\{ \sum_{j=1}^{J} \left(c_j + \sum_{i=1}^{I} w_{ij} y_i \right) h_j \right\} \tag{3.21}$$

When there is no constraint on hidden variables, i.e. they are independent binary variables, the maximization can be factorized and moved inside the sum

$$g^{\text{uc}}_{\max}(\mathbf{y}) = \sum_{j=1}^{J} \max \left\{ c_j + \sum_{i=1}^{I} w_{ij} y_i, 0 \right\} \tag{3.22}$$

The superscript "uc" is short for "unconstrained". This is exactly the same as the "sum" compositional pattern potentials in Eq. 3.17.

When we put a 1-of-J constraint on hidden variables, i.e. forcing $\sum_{j=1}^{J} h_j = 1$, the maximization becomes

$$g^{\text{1of}J}_{\max}(\mathbf{y}) = \max_{1 \le j \le J} \left\{ c_j + \sum_{i=1}^{I} w_{ij} y_i \right\} \tag{3.23}$$

This is exactly the same as the "max" compositional pattern potentials in Eq. 3.18.

3.2.3 Summing out hidden variables in RBMs

The key observation that relates the pattern potentials and RBMs with hidden variables summed out is the following approximation,

$$\max\{x, 0\} \approx \log(1 + \exp(x)) \tag{3.24}$$

This is illustrated in Fig 3.1 (a).

With this approximation, we can rewrite the basis pattern potential in Eq. 3.14 as

$$f(\mathbf{y}) \approx \log\left(1 + \exp\left(c + \sum_i w_i y_i\right)\right) \tag{3.25}$$

On the other hand, summing out hidden variables in an RBM with no constraints on hidden variables,

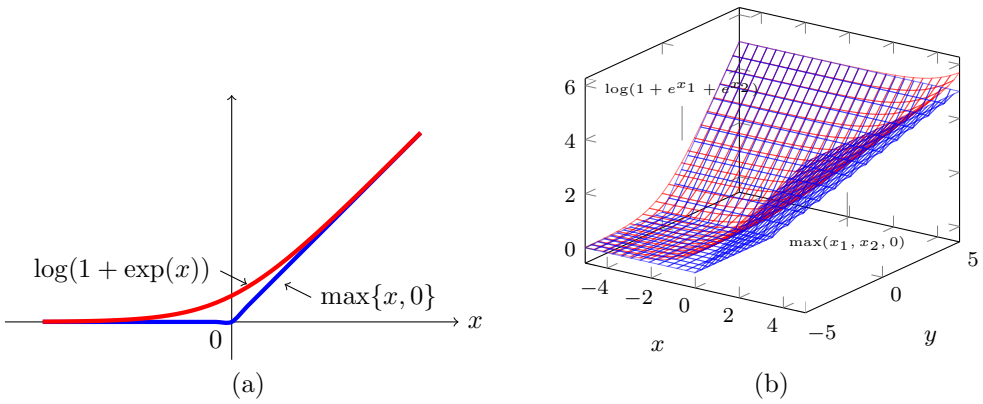


Figure 3.1: (a) $\log(1+\exp(x))$ is a smoothed approximation to $\max\{x, 0\}$; (b) $\log(1+\exp(x_1)+\exp(x_2))$ is a smoothed approximation to $\max\{x_1, x_2, 0\}$.

the marginal distribution becomes

$$p(\mathbf{y}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{I} b_i y_i\right) \prod_{j=1}^{J} \left(1 + \exp\left(c_j + \sum_{i=1}^{I} w_{ij} y_i\right)\right) \tag{3.26}$$

Eq. 3.7 is another equivalent form of this. Therefore the equivalent high order potential induced by summing out the hidden variables is

$$g_{\text{sum}}^{\text{uc}}(\mathbf{y}) = \sum_{j=1}^{J} \log\left(1 + \exp\left(c_j + \sum_{i=1}^{I} w_{ij} y_i\right)\right) \tag{3.27}$$

which is exactly a sum of potentials in the form of Eq. 3.25.

Now we turn to the "max" case. We show that the compositional pattern potentials are equivalent to RBMs with a 1-of-J constraint on hidden variables and hidden variables summed out, up to the following approximation

$$\max\{x_1, x_2, ..., x_J, 0\} \approx \log\left(1 + \sum_{j=1}^{J} \exp(x_j)\right) \tag{3.28}$$

This is a high dimensional extension to Eq. 3.24. The 2-D case is illustrated in Fig 3.1 (b).

We use the definition of "max" compositional pattern potentials in Eq. 3.16, but fix $d_J(\mathbf{y})$ to be 0, to make a constant threshold on the cost.

Then we can subtract constant θ_J from the potential and absorb θ_J into all other θ_j's (with the same change of variable tricks from θ to c) to get

$$f^m(\mathbf{y}) = \max\left\{c_1 + \sum_i w_{i1} y_i, ..., c_{J-1} + \sum_i w_{i,J-1} y_i, 0\right\} \tag{3.29}$$

Using the approximation, this high order potential becomes

$$f^m(\mathbf{y}) \approx \log\left(1 + \sum_{j=1}^{J-1} \exp\left(c_j + \sum_i w_{ij} y_i\right)\right) \tag{3.30}$$

In an RBM with J hidden variables, the 1-of-J constraint is equivalent to $\sum_{j=1}^{J} h_j = 1$. With this constraint, the negative energy (Eq. 3.5) can be transformed into

$$\begin{aligned} -E(\mathbf{y}, \mathbf{h}) &= \sum_{i=1}^{I} b_i y_i + \sum_{j=1}^{J-1} \left(c_j - c_J + \sum_{i=1}^{I} (w_{ij} - w_{iJ}) y_i\right) h_j + \left(c_J + \sum_{i=1}^{I} w_{iJ} y_i\right) \\ &= \sum_{i=1}^{I} (b_i - w_{iJ}) y_i + \sum_{j=1}^{J-1} \left(c_j - c_J + \sum_{i=1}^{I} (w_{ij} - w_{iJ}) y_i\right) + c_J \end{aligned} \tag{3.31}$$

We can therefore use a new set of parameters $b'_i = b_i - w_{iJ}$, $c'_j = c_j - c_J$ and $w'_{ij} = w_{ij} - w_{iJ}$, and get

$$-E(\mathbf{y}, \mathbf{h}) = \sum_{i=1}^{I} b'_i y_i + \sum_{j=1}^{J-1} \left(c'_j + \sum_{i=1}^{I} w'_{ij} y_i\right) h_j \tag{3.32}$$

We ignored the constant c_J because it is cancelled out when we normalize the distribution. Note that now the set of $J-1$ hidden variables can have at most one to be 1, and they can also be all 0, corresponding to the case that the Jth hidden variable is 1.

Summing out $\mathbf{h}$, we get

$$p(\mathbf{y}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{I} b'_i y_i\right) \left(1 + \sum_{j=1}^{J-1} \exp\left(c'_j + \sum_{i=1}^{I} w'_{ij} y_i\right)\right) \tag{3.33}$$

The constant 1 comes from the Jth hidden variable. The equivalent high-order potential for this model is then

$$g^{\text{1of}J}_{\text{sum}}(\mathbf{y}) = \log\left(1 + \sum_{j=1}^{J-1} \exp\left(c'_j + \sum_{i=1}^{I} w'_{ij} y_i\right)\right) \tag{3.34}$$

which has exactly the same form as Eq. 3.30.

3.3 The CHOPP-Augmented CRF

Understanding the equivalence between RBMs and pattern potentials leads us to define a more general potential — Compositional High Order Pattern Potential (CHOPP),

$$f(\mathbf{y}; T) = T \log\left(\sum_{\mathbf{h}} \exp\left(\frac{1}{T}\left(c_j + \sum_i w_{ij} y_i\right) h_j\right)\right) \tag{3.35}$$

where T is a temperature parameter. The sum over $\mathbf{h}$ is a sum over all possible configurations of hidden variables. As did by Schwing et al. [141], introducing a temperature parameter can smoothly interpolate maximization and summation.

Setting $T = 1$, this CHOPP becomes

$$f(\mathbf{y}; 1) = \log \left(\sum_{\mathbf{h}} \exp \left(\sum_{j=1}^{J} \left(c_j + \sum_{i=1}^{I} w_{ij} y_i \right) h_j \right) \right) \tag{3.36}$$

this is the equivalent RBM high order potential with hidden variables summed out. When there is no constraint on $\mathbf{h}$, the above potential becomes

$$f^{\text{uc}}(\mathbf{y}; 1) = \sum_{j=1}^{J} \log \left(1 + \exp \left(c_j + \sum_{i=1}^{I} w_{ij} y_i \right) \right) \tag{3.37}$$

When there is a 1-of-J constraint on $\mathbf{h}$, the above potential is

$$f^{\text{1of}J}(\mathbf{y}; 1) = \log \left(\sum_{j=1}^{J} \exp \left(c_i + \sum_{i=1}^{I} w_{ij} y_i \right) \right) \tag{3.38}$$

Setting $T \to 0$, the CHOPP becomes

$$f(\mathbf{y}; 0) = \max_{\mathbf{h}} \left\{ \sum_{j=1}^{J} \left(c_j + \sum_{i=1}^{I} w_{ij} y_i \right) h_j \right\} \tag{3.39}$$

this is exactly the same as the high order potential induced by an RBM with hidden variables maximized out, and therefore equivalent to composite pattern potentials as shown in Section 3.2.2. When there are no constraints on hidden variables we will get the "sum" composite pattern potentials, while adding a 1-of-J constraint will give us the "max" composite pattern potentials.

Therefore, by using a temperature parameter T, CHOPPs can smoothly interpolate summing out hidden variables (usually used in RBMs) and maximizing out hidden variables (used in Rother et al.[138]). On the other hand, by using extreme sparsity (the 1-of-J constraint), it interpolates the "sum" and "min" composition schemes.

In this section, we introduce how to augment standard pairwise CRF with this type of CHOPPs and describe inference and learning algorithms. We do not enforce any constraint on hidden variables in the following discussion, but it is possible to derive the inference and learning algorithms for the case where we have a soft sparsity or hard 1-of-J constraints on hidden variables. In all the experiments, we used the most basic version of the CHOPP with $T = 1$.

3.3.1 Model

The joint probability of a labeling $\mathbf{y}$ and a vector of binary hidden variables $\mathbf{h}$ given input image $\mathbf{x}$ is defined as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \left\{ \lambda^u \sum_i f_i(y_i|\mathbf{x}) + \sum_k \lambda_k^p \sum_{i,j} f_{ij}^k(y_i, y_j|\mathbf{x}) + \mathbf{b}^\top \mathbf{y} + T \log \left(\sum_{\mathbf{h}} \exp \left(\frac{1}{T} \left(\mathbf{y}^\top \mathbf{W} \mathbf{h} + \mathbf{c}^\top \mathbf{h} \right) \right) \right) \right\} \tag{3.40}$$

where $f_i(y_i|\mathbf{x})$ are unary potentials, $f_{ij}^k(y_i, y_j|\mathbf{x})$ are K different types of pairwise potentials, λ^u and λ_k^p are trade-off parameters for unary and pairwise potentials respectively, and $\mathbf{W}$, $\mathbf{b}$, $\mathbf{c}$ are RBM parameters.

To simplify notation, for a given $\mathbf{x}$ we can denote $\psi^u(\mathbf{y}) = \lambda^u \sum_i f_i(y_i|\mathbf{x})$ for unary potentials, $\psi^p(\mathbf{y}) = \sum_k \lambda_k^p \sum_{i,j} f_{ij}^k(y_i, y_j|\mathbf{x})$ for pairwise potentials, and the total score function

$$f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \psi^u(\mathbf{y}) + \psi^p(\mathbf{y}) + \mathbf{b}^\top \mathbf{y} + T \log \left(\sum_{\mathbf{h}} \exp \left(\frac{1}{T} \left(\mathbf{y}^\top \mathbf{W} \mathbf{h} + \mathbf{c}^\top \mathbf{h} \right) \right) \right) \tag{3.41}$$

with $\boldsymbol{\theta}$ denoting the collection of all the parameters in this model.

$T = 1$ **Special Case** For the special case $T = 1$, the posterior distribution $p(y|x)$ is equivalent to a joint distribution over $\mathbf{y}$ and $\mathbf{h}$, with $\mathbf{h}$ summed out

$$p(\mathbf{y}, \mathbf{h}|\mathbf{x}) \propto \exp \left(\psi^u(\mathbf{y}) + \psi^p(\mathbf{y}) + \mathbf{b}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{W} + \mathbf{c}^\top \mathbf{h} \right) \tag{3.42}$$

Given $\mathbf{y}$, the distribution of $\mathbf{h}$ factorizes, and we have

$$p(h_j = 1|\mathbf{y}, \mathbf{x}) = \sigma \left(c_j + \sum_i w_{ij} y_i \right) \tag{3.43}$$

where σ is the logistic sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$.

Given $\mathbf{h}$, the distribution of $\mathbf{y}$ becomes a pairwise MRF with only unary and pairwise potentials

$$p(\mathbf{y}|\mathbf{h}, \mathbf{x}) \propto \exp \left(\left(\mathbf{b} + \mathbf{W} \mathbf{h} \right)^\top \mathbf{y} + \psi^u(\mathbf{y}) + \psi^p(\mathbf{y}) \right) \tag{3.44}$$

where $(\mathbf{b} + \frac{1}{T} \mathbf{W} \mathbf{h})^\top \mathbf{y} + \psi^u(\mathbf{y})$ is the new unary potential.

Model Variants One way to make this model even more expressive is to make the RBM energy also conditioned on the input image $\mathbf{x}$. The current formulation of CHOPPs is purely unconditional, but knowing some image evidence can help the model determine which pattern should be active. We achieve this by making the hidden biases $\mathbf{c}$ also a function of the input image feature vector $\phi(\mathbf{x})$. The simplest form of this is a linear function $\mathbf{c}(\mathbf{x}) = \mathbf{c}_0 + \mathbf{W}_0^\top \phi(\mathbf{x})$, where $\mathbf{c}_0$ and $\mathbf{W}_0$ are parameters.

Another variant of the current formulation is to make the RBM *convolutional* which entails shrinking the window of image labels $\mathbf{y}$ on which a given hidden unit depends, and devoting a separate hidden unit to each application of one of these feature functions to every possible location in the image [99, 126]. These can be trained by tying together the weights between $\mathbf{y}$ and hidden variables $\mathbf{h}$ at all locations an image. This significantly reduces the number of parameters in the model, and may have the effect of making the CHOPPs capture more local patterns.

3.3.2 Inference

The task of inference is to find the $\mathbf{y}$ that maximize the log probability $\log p(\mathbf{y}|\mathbf{x})$ for a given $\mathbf{x}$. Direct optimization is hard due to the CHOPP which couples all the components of $\mathbf{y}$ together, but we can utilize a variational lower bound:

$$f(\mathbf{y}; T) \geq (\mathbf{c} + \mathbf{W}^\top \mathbf{y})^\top \mathbb{E}_q[\mathbf{h}] + T H(q) \tag{3.45}$$

where $q(\mathbf{h})$ is an arbitrary distribution over $\mathbf{h}$, and $H(q)$ is the entropy of q. Note the temperature parameter T cancels out in the first term. The difference of the left side and the right side is T times

the KL-divergence between q and a distribution p^*

$$p^*(\mathbf{h}|\mathbf{y}) = \frac{\exp\left(\frac{1}{T}(\mathbf{c} + \mathbf{W}^\top \mathbf{y})^\top \mathbf{h}\right)}{\sum_{\mathbf{h}} \exp\left(\frac{1}{T}(\mathbf{c} + \mathbf{W}^\top \mathbf{y})^\top \mathbf{h}\right)}. \tag{3.46}$$

When there is no constraint on $\mathbf{h}$, this is also a factorial distribution. Using this lower bound, we have

$$f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) \geq \psi^u(\mathbf{y}) + \psi^p(\mathbf{y}) + \mathbf{b}^\top \mathbf{y} + (\mathbf{c} + \mathbf{W}^\top \mathbf{y})^\top \mathbb{E}_q[\mathbf{h}] + TH(q) + \text{const} \tag{3.47}$$

We can use the EM algorithm to optimize this lower bound. Starting from an initial labeling $\mathbf{y}$, we alternate the following E step and M step:

In the E step, we fix $\mathbf{y}$ and maximize the bound with respect to q, which is achieved by setting $q(\mathbf{h}) = p^*(\mathbf{h}|\mathbf{y})$. When $T = 1$ this becomes Eq. 3.43; when $T \to 0$, it puts all the mass on one single configuration of $\mathbf{h}$.

In the M step, we fix q and find the $\mathbf{y}$ that maximizes the bound, the relevant terms are

$$\psi^u(\mathbf{y}) + \psi^p(\mathbf{y}) + \left(\mathbf{b} + \mathbf{W}\mathbb{E}_q[\mathbf{h}]\right)^\top \mathbf{y}, \tag{3.48}$$

which is again just a set of unary potentials, so we can use standard optimization methods for pairwise CRFs to find an optimal $\mathbf{y}$; we use graph cuts. If the CRF inference algorithm used in the M step is exact, this algorithm will find a sequence of $\mathbf{y}$'s that monotonically increase the log probability, and is guaranteed to converge.

Note that this is not the usual EM algorithm used for learning parameters in latent variable models. Here all parameters are fixed and we use the EM algorithm to make predictions.

Remark. When there is no sparsity constraint on h, it is possible to analytically sum out the hidden variables, which leads to a collapsed energy function with J high order factors, one for each original hidden unit. It is then possible to develop a linear program relaxation-based inference routine that operates directly on the high order model. We did this but found its performance inferior to the above EM procedure.

3.3.3 Learning

Here we fix the unary and pairwise potentials and focus on learning the parameters in the CHOPP.

For the $T = 1$ case, we can use Contrastive Divergence (CD) [61] to approximately maximize the conditional likelihood of data under our model, which is standard for learning RBMs. However we found that CD does not work very well because it is only learning the shape of the distribution in a neighborhood around the ground truth (by raising the probability of the ground truth and lowering the probability of everything else). In practice, when doing prediction using the EM algorithm on test data, inference does not generally start near the ground truth. In fact, it typically starts far from the ground truth (we use the prediction by a model with only unary and pairwise potentials as the initialization), and the model has not been trained to move the distribution from this region of label configurations towards the target labels.

Instead, we train the model to minimize expected loss which we believe allows the model to more globally learn the distribution. For any image $\mathbf{x}$ and the ground truth labeling $\mathbf{y}^*$, we have a loss

$\ell(\mathbf{y}, \mathbf{y}^*) \geq 0$ for any $\mathbf{y}$ that measures the deviation of $\mathbf{y}$ from $\mathbf{y}^*$. The expected loss is defined as

$$L = \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})\ell(\mathbf{y}, \mathbf{y}^*), \tag{3.49}$$

where $p(\mathbf{y}|\mathbf{x})$ is the model distribution that contains learnable parameters. The expected loss for a dataset is simply a sum over all individual data cases. The following discussion will be for a single data case to simplify notation.

Taking the derivative of the expected loss with respect to model parameter θ, which can be $\mathbf{b}$, $\mathbf{c}$ or $\mathbf{W}$ ($\mathbf{c}_0$ and $\mathbf{W}_0$ as well if we use the conditioned CHOPPs), we have

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{\theta}} &= \sum_{\mathbf{y}} \frac{\partial p(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \ell(\mathbf{y}, \mathbf{y}^*) \\
&= \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \frac{\partial \log p(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \ell(\mathbf{y}, \mathbf{y}^*) \\
&= \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \frac{\partial}{\partial \boldsymbol{\theta}} \left[f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) - \log \sum_{\mathbf{y}} \exp(f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})) \right] \ell(\mathbf{y}, \mathbf{y}^*) \\
&= \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \left[\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} - \frac{\sum_{\mathbf{y}} \exp(f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})) \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}}}{\sum_{\mathbf{y}} \exp(f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}))} \right] \ell(\mathbf{y}, \mathbf{y}*) \\
&= \mathbb{E}_{\mathbf{y}} \left\{ \left[\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} - \mathbb{E}_{\mathbf{y}} \left[\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \right] \right] \ell(\mathbf{y}, \mathbf{y}^*) \right\} \\
&= \mathbb{E}_{\mathbf{y}} \left[\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \ell(\mathbf{y}, \mathbf{y}^*) \right] - \mathbb{E}_{\mathbf{y}} \left[\mathbb{E}_{\mathbf{y}}[\ell(\mathbf{y}, \mathbf{y}^*)] \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \right] \\
&= \mathbb{E}_{\mathbf{y}} \left[(\ell(\mathbf{y}, \mathbf{y}^*) - \mathbb{E}_{\mathbf{y}}[\ell(\mathbf{y}, \mathbf{y}^*)]) \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \right]
\end{aligned} \tag{3.50}$$

where $\mathbb{E}_{\mathbf{y}}[.]$ is the expectation under $p(\mathbf{y}|\mathbf{x})$. For the CHOPP-augmented CRF model, the derivative $\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ is easy to compute for parameters λ^u and λ^p_k, for the CHOPP parameters, we have

$$\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = -\mathbb{E}_{p^*} \left[\frac{\partial E(\mathbf{y}, \mathbf{h})}{\partial \boldsymbol{\theta}} \right] \tag{3.51}$$

where p^* is the distribution derived in Eq. 3.46, and $E(\mathbf{y}, \mathbf{h})$ is the standard RBM energy function.

Using a set of samples $\{\mathbf{y}^n\}_{n=1}^N$ from $p(\mathbf{y}|\mathbf{x})$, we can compute an unbiased estimation of the gradient

$$\frac{\partial L}{\partial \theta} \approx \frac{1}{N-1} \sum_{n} \left(\ell(\mathbf{y}^n, \mathbf{y}^*) - \frac{1}{N} \sum_{n'} \ell(\mathbf{y}^{n'}, \mathbf{y}^*) \right) \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x})}{\partial \boldsymbol{\theta}} \tag{3.52}$$

This gradient has an intuitive explanation: if a sample has a loss lower than the average loss of the batch of samples, then we should reward it by raising its probability, and if its loss is higher than the average, then we should lower its probability. Therefore even when the samples are far from the ground truth, we can still adjust the relative probabilities of the samples. In the process, the distribution is shifted in the direction of lower loss.

For the $T = 1$ case, we sample from the joint distribution $p(\mathbf{y}, \mathbf{h}|\mathbf{x})$ using standard block Gibbs sampling and discard $\mathbf{h}$ to get samples from $p(\mathbf{y}|\mathbf{x})$. We also use several persistent Markov chains for each image to generate samples, where in the first iteration of learning each chain is initialized at the

same initialization as is used for inference. The model parameters are updated after every sampling step. For the other choices of T, it is not easy to get samples from $p(\mathbf{y}|\mathbf{x})$, but we can sample from $p^*(\mathbf{h}|\mathbf{y})$ and $p(\mathbf{y}|\mathbf{h},\mathbf{x})$ alternatively, as if we are running block Gibbs sampling. The properties of the samples that result from this procedure is an interesting topic for future research.

Remark This expected loss minimization learning algorithm is very general, and can be applied to almost any probabilistic model. Notably, in this learning algorithm we do not assume any decompositionality or even differentiability of the loss function $\ell(\mathbf{y},\mathbf{y}^*)$, which means we can optimize *any arbitrary computable loss function*, including non-differentiable losses. In the experiments, we use this method to directly optimize the Intersection over Union (IoU) score for segmentation.

3.4 Experiments

We evaluate our CHOPP-augmented CRF on synthetic and real data sets on the task of image segmentation. The settings for synthetic data sets will be explained later. For all the real datasets, we extracted a 107 dimensional descriptor for each pixel in an image by applying a filter bank, which includes color features (RGB and Lab, 6 features), responses to Gabor filters (5 filter frequencies and 4 filter orientations, which gives us $5 \times 4 \times 2 = 40$ features), Leung-Malik filters (48 features) and Schmid filters (13 features). We used the implementation of Leung-Malik and Schmid filterbank from `http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html`. All the filters are applied to the grey scale image. We trained a 2-layer (1 layer of hidden units) neural network classifier using these descriptors as input and use the log probability of each class for each pixel as the unary potentials.

For pairwise potentials, we used a standard 4-connected grid neighborhood and the common Potts model, where $f_{ij}(y_i,y_j|\mathbf{x}) = p_{ij}\mathbf{I}[y_i \neq y_j]$ and p_{ij} is a penalty for assigning different labels for y_i and y_j. Three different ways to define p_{ij} yield three pairwise potentials:

(1) Set p_{ij} to be constant, this would enforce smoothing for the whole image;

(2) Set p_{ij} to incorporate local contrast information by computing RGB differences between pairs of pixels as in [14], where $p_{ij} = \exp\left(-\frac{\|I_i - I_j\|^2}{2\sigma^2}\right)$, I_i, I_j are RGB values for the two pixels and σ is a parameter controlling the sensitivity to contrast;

(3) Set p_{ij} to represent higher level boundary information given by Pb boundary detector [113], more specifically, we define $p_{ij} = -\max\{\log Pb_i, \log Pb_j\}$ where Pb_i and Pb_j are the probability of boundary for pixel i and j.

For each dataset, we hold out a part of the training data to make a validation set, and we use it to choose hyper parameters, e.g. the number of iterations to run in training. We choose the model that performs the best on the validation set and report its performance on a separate test set.

For all experiments, the performance of the models are evaluated using the Intersection over Union (IoU) score.

3.4.1 Data Sets & Variability

Throughout the experiments, we use six synthetic and three real world data sets. In the experiments, we found that the benefit of using CHOPPs is related to the amount of variability in data. To explore data set variability in a controlled fashion, we generated a series of increasingly variable synthetic data sets.

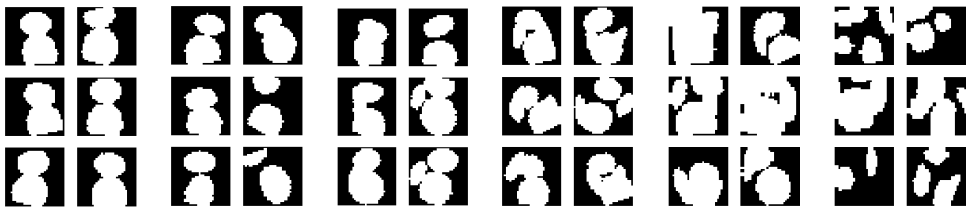


Figure 3.2: Randomly sampled examples from synthetic data set labels. Hardness increases from left to right. Quantitative measures of variability using $K = 32$ are reported in the bottom row. Variabilities of Horse, Bird, and Person data sets are 0.176, 0.370, and 0.413.

The datasets are composed of between 2 and 4 ellipses with centers and sizes chosen to make the figures look vaguely human-like (or at least snowman-like). We then added noise to the generation procedure to produce a range of six increasingly difficult data sets, which are illustrated in Fig. 3.2 (top row). To generate associated unary potentials, we added Gaussian noise with standard deviation 0.5. In addition, we added structured noise to randomly chosen 5-pixel diameter blocks.

The real world data sets come from two sources: first, we use the Weizmann horses and resized all images as well as the binary masks to 32×32; second, we use the PASCAL VOC 2011 segmentation data [37] to construct a bird and a person data set. For these, we take all bounding boxes containing the target class and created a binary segmentation of the inside of the bounding box, labeling all pixels of the target class as 1, and all other pixels as 0. We then transformed these bounding boxes to be of size 32×32. This gives us a set of silhouettes that preserve the challenging aspects of modeling shape in a realistic structured output setting. Some images from the three real data sets can be seen in Figures 3.4, 3.5, 3.6 and 3.7.

The two PASCAL datasets are challenging due to variability in the images and segmentations, while the number of images is quite small (214 images for birds and 1169 for person), especially compared to the settings where RBM models are typically used. When we are only training the trade-off parameters, this is not a major problem, because the number of parameters is small. But here we also train internal parameters of high order potentials, which require more data for training to work well. To deal with this problem, we generated 5 more bounding boxes for each original bounding box by randomly shifting coordinates by a small amount. We also mirrored all images and segmentations. This augmentation gives us 12 times as many training examples.

For each data set, we then evaluated variability. To do so, we propose a measure inspired by the learning procedure suggested by Rother et al. [138]. First, cluster segmentations using K-means clustering with Euclidean distance as the metric. Then for each cluster and pixel, compute the fraction of cases for which the pixel is on across all instances assigned to the cluster. This yields q_{ij}^k, the probability that pixel ij is assigned label 1 given that it comes from an instance in cluster k. Now define the within cluster average entropy $H^k = -\frac{1}{D_v} \sum_{ij} \left(q_{ij}^k \log q_{ij}^k + (1 - q_{ij}^k) \log(1 - q_{ij}^k) \right)$, where D_v is the number of pixels in the image. Finally, the variability measure is a weighted average of within cluster average entropies: $V_K = \sum_{k=1}^{K} \mu_k H^k$, where μ_k is the fraction of data points assigned to cluster k. We found $K = 32$ to work well and used it throughout. We found the quantitative measure matches intuition about the variability of data sets as shown in Fig. 3.2.

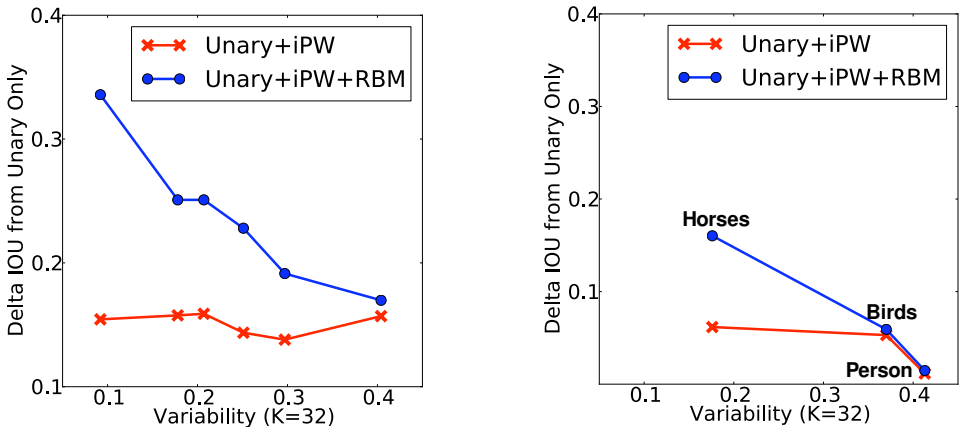


Figure 3.3: Results on (left) synthetic and (right) real data showing test Intersection over Union (IoU) scores as a function of data set variability. The y-axis is the difference relative to Unary Only model. Note that these results are for the pretrained RBM model.

3.4.2 Performance vs. Variability

Next we report results for a pre-trained RBM model added to a standard CRF (denoted RBM). Here, we learn the RBM parameters offline on the labels using PCD [160] and set tradeoff parameters so as to maximize accuracy on the training set. We compare the Unary Only model to the Unary+Pairwise model and the Unary+Pairwise+RBM model. Predictions for the Unary+Pairwise models are made by graph cuts, while for the Unary+Pairwise+RBM model the inference algorithm introduced in Section 3.3.2 is used. Pairwise terms are image dependent, meaning that all 3 types of pairwise potentials are used, which is denoted by iPW. Fig. 3.3 shows the results as a function of the variability measure described in the previous section. On the y-axis, we show the difference in performance between the Unary+iPW and Unary+iPW+RBM models versus the Unary Only model. In all but the Person data set, the Unary+iPW model provides a consistent benefit over the Unary Only model. For the Unary+iPW+RBM model, there is a clear trend that as the variability of the data set increases, the benefit gained from adding the RBM declines.

3.4.3 Improving on Highly Variable Data

We now turn our attention to the challenging real data sets of Birds and Person and explore methods for improving the performance of the RBM component when the data becomes highly variable.

Training with Expected Loss The first approach to extending the pretrained RBM+CRF model that we consider is to jointly learn the internal potential parameters $\mathbf{W}, \mathbf{c}$ and $\mathbf{b}$. Initial experiments with standard contrastive divergence learning on the Horse data led to poor performance, as the learning was erratic in the first few iterations and then steadily got worse during training. So here we focus on the offline pretraining and the expected loss training described in Section 3.3.3. We use 2 sampling chains[3] for each image and use the validation set to do early stopping. The learning rate is fixed and chosen from $\{10, 1, 0.1, 0.01\}$ (the gradients are quite small so we tried some large learning rates here) so that it is small enough to avoid erratic behavior and big enough to make significant updates of the weights

[3]We tried 10 sampling chains for each image as well, but it didn't give us any significant performance boost over 2 sampling chains and it was much slower.

Method	Horse IoU	Bird IoU	Person IoU
Unary Only	0.5119	0.5055	0.4979
Unary+iPW	0.5736	0.5585	0.5094
Unary+iPW+RBM	0.6722	0.5647	0.5126
Unary+iPW+jRBM	**0.6990**	**0.5773**	**0.5253**

Table 3.2: Expected loss test results. RBM is a pretrained RBM. jRBM is jointly trained using expected loss.

in reasonable time. We denote the resulting RBM models as jRBM to indicate joint training. Results comparing these approaches on the three real data sets are given in Table 3.2, with Unary+iPW results given as a baseline. We see that training with the expected loss criterion improves performance across the board.

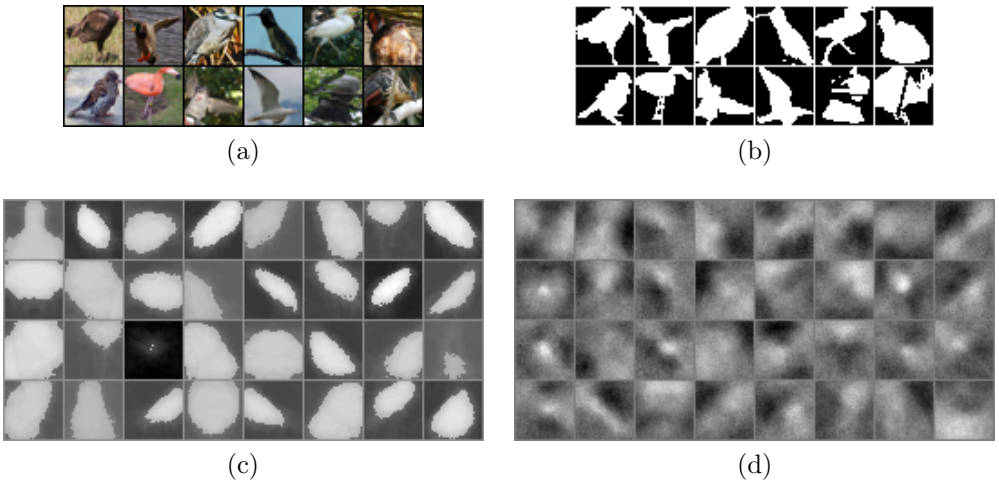
Image-dependent Hidden Biases Here, we consider learning image-dependent hidden biases as described in Section 3.3.1 (modeling hidden biases $\mathbf{c}$ as a linear function of some image feature $\phi(\mathbf{x})$). As inputs, we use the learned unary potentials and the response of the Pb boundary detector [113], both downsampled to be of size 16×16. We jointly learned the RBM internal parameters using the IoU expected loss, as this gave the best results in the previous experiment. We refer to these jointly trained, image-dependent RBMs with ijRBM. Results are shown in Table 3.3. For comparison, we also train Unary+Pairwise models with a image-independent pairwise potentials (PW) along with the standard image-dependent pairwise potentials (iPW). In the Bird data, we see that the image-specific information helps the ijRBM similarly as how image-dependent pairwise potentials improve over image-independent pairwise potentials. In the Person data, the gains from image-dependent information are minimal in both cases.

Convolutional Structures Our final experiment explores the convolutional analog to the RBM models discussed in Section 3.3.1 . Unfortunately, we were unable to achieve good results. We tried two variants: (a) a vanilla pre-trained convolutional RBM, and (b) a pre-trained convolutional RBM with conditional hidden biases as described in Section 3.3.1. We tried two different patch sizes (8×8, 12×12) and tiled the images densely. Though the conditional variant outperformed the unconditional variant, overall results were discouraging—performance was not even as good as the simple Unary+Pairwise model. This is surprising because a convolutional RBM should in theory be able to easily represent pairwise potentials, and convolutional RBMs have fewer parameters than their global counterparts, so overfitting should not be an issue. We believe the explanation for the poor performance is that learning methods for convolutional RBMs are not nearly as evolved as methods for learning ordinary RBMs, and thus the learning methods that we have at our disposal do not perform as well. On the bright side, this can be seen as a challenge to overcome in future work. A few methods developed for tiled convolutional (not fully convolutional) RBMs achieved good results modeling textures [75], which shows some potential that this may be a good way to go.

Composition Schemes We qualitatively compare patterns learned for the "min" composition approach presented in [138] using k-means versus the patterns learned by a simple pre-trained RBM, which are appropriate for "sum" composition. While a quantitative comparison that explores more degrees of freedom offered by CHOPPs is a topic for future work, we can see in Fig. 3.4 that the filters learned are very different. As the variability of the data grows, we expect the utility of the "sum" composition scheme to increase.

Method	Bird IoU	Person IoU
Unary+PW	0.5321	0.5082
Unary+iPW	0.5585	0.5094
Unary+iPW+jRBM	0.5773	**0.5253**
Unary+iPW+ijRBM	**0.5858**	**0.5252**

Table 3.3: Test results using image-specific hidden biases on the high variability real data sets. PW uses image-independent pairwise potentials, and iPW uses image-dependent pairwise potentials. jRBM is jointly trained but image independent. ijRBM is jointly trained and has learned image-dependent hidden biases.



(a) (b) (c) (d)

Figure 3.4: (a) Images from Bird data set. (b) Ground truth labels. (c) Patterns learned by clustering-style approach of [138]. (d) Patterns learned by compositional-style learning used in this paper.

3.4.4 Qualitative Prediction Results

Some example segmentations for horse, bird and person data sets are given in Fig. 3.5, Fig. 3.6 and Fig. 3.7.

3.5 Discussion

In this chapter, we began by precisely mapping the relationship between pattern potentials and RBMs, and generalizing both to yield CHOPPs, a class of high order potential that includes both as special cases. The main benefit of this mapping is that it allows the leveraging of complementary work from two mostly distinct communities. First, it opens the door to the large and highly evolved literature on learning RBMs. These methods allow efficient and effective learning when there are hundreds or thousands of latent variables. There are also well-studied methods for adding structure over the latent variables, such as sparsity. Conversely, RBMs may benefit from the highly developed inference procedures that are more common in the structured output community e.g. those based on linear programming relaxations. Also interesting is that pairwise potentials provide benefits that are reasonably orthogonal to those offered by RBM potentials. In particular, a vanilla application of RBMs is unable to capture the image-dependent pairwise potential strengths that are common in image segmentation.

Empirically, our work emphasizes the importance of data set variability in the performance of these

(a) Best (b) Average (c) Worst

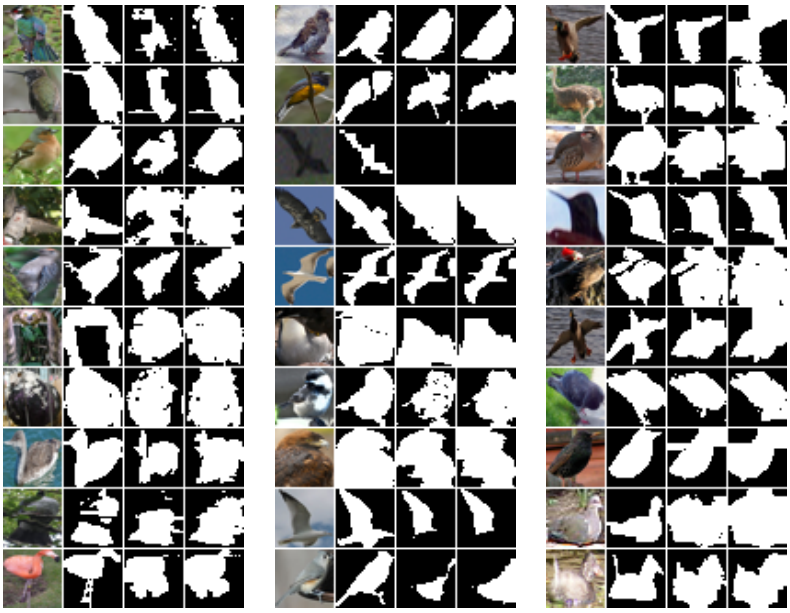
Figure 3.5: Prediction results on horse data set. The three categories best, average and worst are measured by the improvement of Unary+Pairwise+RBM over Unary+Pairwise. Each row left to right: original image, ground truth, Unary+Pairwise prediction, Unary+Pairwise+RBM prediction.

methods. It is possible to achieve large gains on low variability data, but it is a challenge on high variability data. Our proposed measure for quantitatively measuring data set variability is simple but useful in understanding what regime a data set falls in. This emphasizes that not all "real" data sets are created equally, as we see moving from Horse to Bird to Person. While we work with small images and binary masks, we believe that the high variability data sets we are using preserve the key challenges that arise in trying to model shape in real image segmentation applications. Note that it would be straightforward to have a separate set of shape potentials per object class within a multi-label segmentation setting.

To attain improvements in high variability settings, more sophisticated methods are needed. Our contributions of training under an expected loss criterion and adding conditional hidden biases to the model yield improvements on the high variability data. There are other architectures to explore for making the high order potentials image-dependent. In future work, we would like to explore multiplicative interactions [116], and other forms of weight sharing that enforce symmetries that we expect to find in segmentation data.

The convolutional approach appears promising, but it did not yield improvements in our initial experiments, which we attribute to the relatively nascent nature of convolutional RBM learning techniques. A related issue that should be explored in future work is the issue of sparsity in latent variable activations. We showed in Section 3.2 that this sparsity can be used to control the type of compositionality employed by the model (extreme 1-of-J sparsity vs. no sparsity). An interesting direction for future work is exploring sparse variants of RBMs, which sit in between these two extremes, and other forms of structure over latent variables like in deep models.

At the same time as our work, [69] also used RBMs in CRFs for image segmentation problems, but they trained the model based on a CD-style algorithm, while we used the expected loss minimization,



(a) Best (b) Average (c) Worst

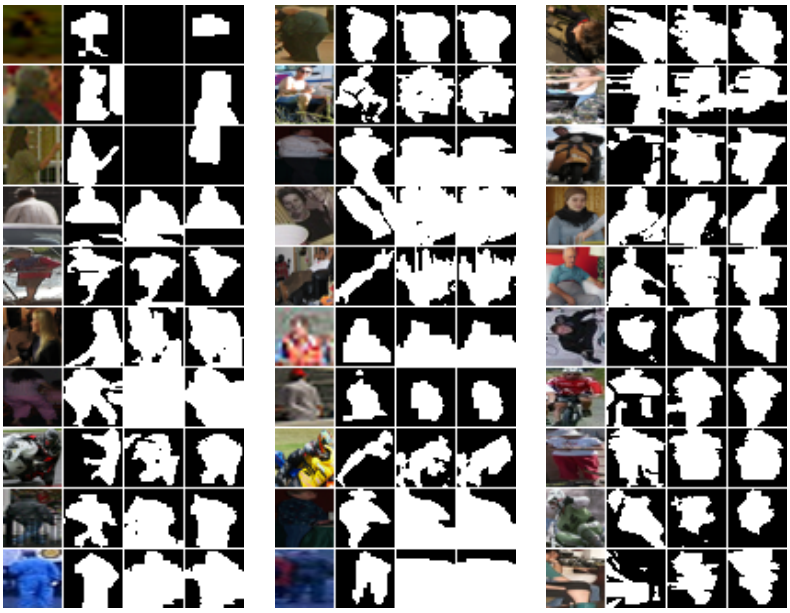
Figure 3.6: Prediction results on bird data set.

which can better incorporate task loss information into learning the model. Training with this expected loss formulation is especially interesting and can be applied to train models to optimize any computable loss function without requiring it to be differentiable. This expected loss minimization algorithm has a close similarity to the REINFORCE algorithm [176] used in reinforcement learning, even though we were not aware of this connection when we did this work. Compared to the commonly used REINFORCE algorithm, our expected loss minimization update proposed in Eq. 3.50 has a few special properties: (1) the average loss $\mathbb{E}_{\mathbf{y}}[\ell(\mathbf{y}, \mathbf{y}^*)]$ automatically provides a baseline; (2) in the Monte Carlo estimate, we only need to have access to the unnormalized log probability, rather than the log probability itself. It would be interesting to see this kind of expected loss minimization be applied to other problems and maybe even reinforcement learning problems.

Later on, Yang et al. [178] extended our work to a max-margin formulation, and also tried to use multiple layers of hidden variables instead of a single hidden layer as in a RBM model.

From our empirical experience, these pattern-like potentials work best when the patterns can be reliably learned and represented by pattern templates. In order to achieve this, we need a good amount of training data to learn these patterns, which is a challenge because for structured output problems acquiring the output labels is a lot harder than for simple prediction problems, and our models of patterns are built directly on these expensive labels. This suggests that some semi-supervised or unsupervised learning methods may be potentially very beneficial for such models.

On the other hand, the current representation of the patterns are based on templates, which has a lot of parameters and can be hard to learn with limited amount of data. To build models on high resolution images and on high variability data, using these full templates is not practical, suggesting that convolutional models are necessary. From another angle, low level and local patterns may be more reliably captured by pattern templates because of the amount of possible patterns is smaller, while at a more global scale the level of variability may be too high to be modeled with pattern templates efficiently.



(a) Best (b) Average (c) Worst

Figure 3.7: Prediction results on person data set.

Ideally, a convolutional and multi-layer pattern model is more desirable than a single layer model as we used in our work.

The pattern models are essentially priors about what a desirable labeling should look like. These models can be learned directly on the labels in a way similar to unsupervised learning, as these models are learned to capture which labels are likely and which are not, rather than predicting something for a given input. Given the recent progress in unsupervised learning models, especially the variational autoencoders [72], it is interesting to see how well these models can do for this task. The variational autoencoder model has recently achieved many impressive results on modeling image data and have been shown to be better than RBM based models on these tasks. The pattern modeling problem may benefit from using these new developments in unsupervised learning.

A recent work by Belanger and McCallum [6] provides an alternative approach for modeling patterns in structured outputs. In this work, instead of using RBM-style models as the prior, the authors proposed to directly use a feed-forward neural net that maps the input and output pair directly to an energy value. The benefit of this approach is that the neural net can be more expressive than an RBM. However, inference in this model is harder, and various relaxations have to be used as neural nets only work in the continuous domain. Overcoming these challenges and using more structured neural networks to model the energy functions is a promising direction for future work.

Chapter 4

Semi-Supervised Learning with High-Order Regularization

Supervised learning of structured models require a large amount of labeled data. For structured output problems in particular, due to the complexity of the outputs, obtaining accurate labels requires considerably more effort than for standard classification or regression tasks. As a result, while large classification datasets, such as ImageNet [29], contain millions of labeled examples, the size of typical datasets for structured prediction problems like image segmentation are orders of magnitudes smaller. On the other hand, large amounts of unlabeled data are typically very easy to obtain.

This combination of difficulty to obtain labeled examples for structured prediction problems, with abundant unlabeled data makes semi-supervised learning (SSL) especially worth exploring. However, SSL is challenging for structured prediction because the complex high dimensional output space makes a lot of operations intractable.

A dominant approach to SSL is to use unlabeled data to regularize the model by ensuring that its predictions on unlabeled data are consistent with some prior beliefs. For example, entropy regularization [98] and low density separation [185] regularize the model so that it makes confident predictions on unlabeled data. Graph-based methods [2, 151], on the other hand, regularize the model to make similar predictions for unlabeled data close on a graph.

Posterior regularization (PR) [40] has been introduced as a general framework to incorporate prior constraints about predictions into structured prediction models. A version of it has also been applied to graph-based SSL for sequence labeling [57]. In PR, constraints are specified as regularizers on posterior distributions, and a decomposition technique is used to make the optimization tractable for structured outputs.

In this chapter, we propose a new method for semi-supervised structured output learning based on SSVMs, that allows regularizers to be defined directly on the predictions of the model for unlabeled data, instead of using the posterior distribution as a proxy. This makes it possible to specify a range of regularizers that are not easy to define on distributions, including those involving loss functions and cardinality of outputs.

One advantage of the SSVM framework is that at test time we typically only want to produce the most likely output, which is generally easier than marginal inference in probabilistic frameworks. For example, in image segmentation, MAP inference can be done efficiently on graphs with submodular

pairwise potentials using powerful discrete optimization techniques like graph cuts, which is key to the success of many segmentation methods. However, marginal inference is intractable and even hard to approximate due to the extremely loopy structure of the graph. Therefore while most of the previous work on SSL studied sequences, our new framework is especially suitable for structured outputs beyond 1-D sequences.

In this chapter we also explore the relationship between our method and PR. We show that the two approaches are actually very closely related: our framework and PR optimize two special cases of the same objective function for some general settings. This connection opens a range of new possibilities of designing and analyzing frameworks for incorporating prior constraints into the model.

We then demonstrate the new framework with an application to graph-based SSL for image segmentation. In graph-based SSL, an important issue is to choose a proper similarity metric in the output space. We utilize the loss function, which offers a natural similarity metric in the output space, as the metric in our formulation.

4.1 Related Work

The earliest work on SSL dates back to the study of the wrapper method known as self-training, in the 1960s, e.g., [142]. Self-training iteratively uses the predictions of the model on unlabeled data as true labels to retrain the model. Because of its heuristic nature, this method is hard to analyze.

A wide range of SSL methods have been developed for classification problems to date [125, 67, 46, 184, 182, 7, 10]; see [183] and [23] for excellent surveys and additional references.

Some researchers have adapted these methods to structured output problems. These methods generally fall into one of the following categories:

(1) **Co-training**, which iteratively uses the predictions made by models trained on different views of the same data to label the unlabeled set and update the model using the predicted labels [15]. The applicability of this method is limited due to the requirement of multi-view data.

(2) **Generative models**, which use unlabeled data to help learning a model of the joint input-output distribution $p(\mathbf{x}, \mathbf{y})$. While having some early success for classification problems [125], generative models make strong assumptions about the data and have to date achieved limited success on structured output problems.

(3) **Low density separation based methods**, which encourage confident predictions on unlabeled data. This translates to low entropy of the output posterior distribution in a probabilistic modeling framework [98], and large margin for methods in a max-margin framework [185]. A combined objective is optimized to minimize the sum of the task loss on the labeled data and a separation regularizer on the unlabeled data.

(4) **Graph based methods**, which construct a graph that connects examples that are nearby in the input space, and then encourage the predictions by the model for pairs of connected examples to be close as well. Most of the work in this category deals with sequence labeling problems. Altun et al. [2] uses a graph on parts of $\mathbf{y}$ to derive a graph-regularized kernel which is used in a max-margin framework. Unlike our framework described below, this approach is not able to incorporate other high order regularizers. This work can be thought of as a special case of our framework where

the graph regularizer is defined on parts of $\mathbf{y}$, but our framework also allows the use of higher order regularizers and our optimization method does not require the inversion of a kernel matrix. Subramanya et al. [151] proposes a semi-supervised CRF that infers labels for unlabeled data by propagation on a graph of parts, and then retrains the model using the inferred labels. Finally, Vezhnevets et al. [164] proposes a graph-based method for semi-supervised image segmentation, which utilizes unlabeled examples in learning by inferring labels for them based on a graph defined on image superpixels. While the authors suggest to use the graph also at test time, we only use the model itself to make predictions.

Recently, other general frameworks for SSL in structured output problems have been defined that can be viewed as graph-based. Posterior regularization (PR) [40] is a framework to incorporate constraints on structured probabilistic models through regularizers defined on posterior distributions. He et al. [57] applies this general PR framework to graph-based SSL also using a CRF model. PR is closely related to our framework: we show in Section 4.3 that the two frameworks are optimizing special cases of the same objective. Constraint Driven Learning (CODL) [22] and Generalized Expectation Criteria [114] are two other notable frameworks for incorporating constraints into the model. [107] provides an interesting discussion about supervision through specifying constraints versus supervision through providing training examples. Another related work [167] discussed ways to apply non-local constraints on marginals of a probabilistic model, which solves a similar problem to ours in the continuous domain.

A separate but related line of research is the study of transfer learning or domain adaptation [130], where most of the labeled data comes from a source domain and task performance is evaluated in a different target domain, typically with little labeled data available. We explore some domain adaptation settings in our experiments presented in Section 4.4.

4.2 Formulation

4.2.1 High Order Regularized SSL

In an SSL setting, we have a set of unlabeled data $D_U = \{\mathbf{x}^j\}_{j=L+1}^{L+U}$ in addition to the labeled data $D_L = \{(\mathbf{x}^i, \mathbf{y}^i)_{i=1}^{L}\}$. Our objective for learning is composed of a loss defined on labeled data, and a regularizer defined directly on predictions of the model on unlabeled data:[1]

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \frac{1}{L}\sum_{i=1}^{L} \ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}) + R\left(\{\mathbf{y}^j\}_{j=L+1}^{L+U}\right) \\ \text{s.t.} \quad & \mathbf{y}^j = \operatorname*{argmax}_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}), \quad \forall j \geq L+1 \end{aligned} \tag{4.1}$$

In this formulation, ℓ is a training loss function such as the structured hinge loss defined in Eq. 2.40, $F_{\boldsymbol{\theta}}$ is the scoring function, R is the (high order) regularizer, and the constraints force $\{\mathbf{y}^j\}_{j=L+1}^{L+U}$ to be predictions of the model for unlabeled data.

R specifies prior constraints about the predictions on unlabeled data. A *high-order* regularizer is one that imposes constraints on sets of output elements rather than independently on each element. One example of a high-order R is the cardinality regularizer, where $R(\mathbf{Y}_U)$ is a function of $\mathbf{1}^\top \mathbf{Y}_U$, and

[1] Here we are ignoring data independent regularizers, e.g., L1 and L2, in this formulation for simplicity, but it is straightforward to incorporate them into the model.

the vector $\mathbf{Y}_U$ is defined as a shorthand for the concatenation of all $\mathbf{y}^j$'s for $j \geq L+1$. For example, in a part-of-speech NLP task, this could refer to the number of words labeled as verbs, while in an image segmentation task it could refer to the number of pixels labeled as foreground. This is useful to encourage the predicted labels to have similar count statistics as the labeled data. As observed in many previous papers, e.g., [184, 172], enforcing this type of constraint is important for imbalanced datasets. In Section 4.2.2, we describe a graph based regularizer R and its combination with cardinality regularizers. A variety of other high-order regularizers, e.g., [165, 76, 158, 22, 19], have been defined in various structured output settings.

Minimizing the objective in Eq. 4.1 is difficult due to the hard constraints that make R a complicated and possibly non-continuous function of $\boldsymbol{\theta}$. To solve this difficulty, we utilize some relaxations of the hard constraints.

We observe that these constraints are equivalent to the following when the maximum is unique,

$$F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}^j) = \max_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}), \quad \forall j \geq L+1. \tag{4.2}$$

Since we have $\max_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}) \geq F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}^j)$ for all $\mathbf{y}^j$, the amount of constraint violation can be measured by the difference $\max_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}) - F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}^j)$. We therefore replace the constraints by a term in the objective that penalizes constraint violation,

$$\min_{\boldsymbol{\theta}, \mathbf{Y}_U} \quad \frac{1}{L} \sum_{i=1}^{L} \ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}) + R(\mathbf{Y}_U) + \frac{\mu}{U} \sum_{j=L+1}^{L+U} \left[\max_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}) - F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}^j) \right] \tag{4.3}$$

where $\mu \geq 0$ measures the tolerance of constraint violation. When $\mu \to +\infty$, this is equivalent to Eq. 4.1; when $\mu < +\infty$, this becomes a relaxation of Eq. 4.1, where $\mathbf{Y}_U$ can be different from the predictions made by the model. This relaxation decouples $\boldsymbol{\theta}$ from R and makes it possible to optimize the objective by iterating two steps, alternatively fixing $\boldsymbol{\theta}$ or $\mathbf{Y}_U$ and optimize over the other, where both steps are easier to solve than Eq. 4.1:

Step 1. Fix $\boldsymbol{\theta}$ and optimize over $\mathbf{Y}_U$. The optimization problem becomes

$$\min_{\mathbf{Y}_U} R(\mathbf{Y}_U) - \frac{\mu}{U} \sum_{j=L+1}^{L+U} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}^j), \quad \text{or equivalently} \quad \max_{\mathbf{Y}_U} \frac{\mu}{U} \sum_{j=L+1}^{L+U} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}^j) - R(\mathbf{Y}_U) \tag{4.4}$$

This step infers labels for those unlabeled examples, based on both the current model and the regularizer. This is a MAP inference problem, and the hard part is to handle the high-order regularizer $R(\mathbf{Y}_U)$. A wide range of methods have been developed for computing MAP in models with high-order potentials [165, 76, 158, 157]. We discuss the approach for our loss-based graph regularizer and cardinality regularizers in more detail in Section 4.2.2.

Step 2. Fix $\mathbf{Y}_U$ and optimize over $\boldsymbol{\theta}$. The optimization problem becomes

$$\min_{\boldsymbol{\theta}} \frac{1}{L} \sum_{i=1}^{L} \ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}) + \frac{\mu}{U} \sum_{j=L+1}^{L+U} \left[\max_{\mathbf{y}} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}) - F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}^j) \right] \tag{4.5}$$

This step updates the model using both the labeled data and the labels inferred from Step 1 for unlabeled data. Note that the last term is very close to $\ell(\mathbf{x}^j, \mathbf{y}^j, \boldsymbol{\theta})$ defined in Eq. 2.40 except that the task loss Δ is not used as in the loss-augmented inference. This optimization is no harder than optimizing a fully

supervised model, which can be solved by methods such as subgradient descent, see e.g. Section 2.4.

Thus our learning algorithm proceeds by iteratively solving the optimization problems in Eq. 4.4 and Eq. 4.5.

4.2.2 Graph-Based SSL for Image Segmentation

In this section we describe an application of the proposed framework to graph-based SSL for binary segmentation, but we note that our method can be easily extended to multi-class segmentation. Graph-based SSL uses a graph so constructed that examples close on this graph should have similar outputs. The model is then regularized by this graph to make predictions that are smooth on it. Here we assume the graph is represented by edge weights s_{ij} which measures the similarity between example i and j, and the two examples are connected only when $s_{ij} > 0$.

Choosing a proper output similarity metric is important for graph-based SSL methods. For classification, most graph-based methods define this similarity as the squared difference of two posterior distributions [184, 182]. For structured prediction, [151, 57] follow this approach but use marginal distributions over parts of output in the squared difference.

However, structured output problems have a natural similarity metric in the output space, defined by the task loss function Δ. For probabilistic models, it is not easy to incorporate the loss function into the similarity metric. But our framework allows the use of loss functions in the regularizer R.

We define the graph regularizer

$$R_G(\mathbf{Y}_U) = \lambda \sum_{i,j:s_{ij}>0} s_{ij}\Delta(\mathbf{y}^i, \mathbf{y}^j) \tag{4.6}$$

where the sum is over all edges in the graph, connecting both labeled and unlabeled examples, and λ is a weight factor. This regularizer requires $\mathbf{y}^i$ and $\mathbf{y}^j$ to be close when s_{ij} is large.

To use this regularizer in our framework, we need to solve the MAP inference problem in Step 1 of the algorithm:

$$\max_{\mathbf{Y}_U} \frac{\mu}{U} \sum_{j=L+1}^{L+U} F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y}) - \lambda \sum_{i,j:s_{ij}>0} s_{ij}\Delta(\mathbf{y}^i, \mathbf{y}^j). \tag{4.7}$$

Here each $F_{\boldsymbol{\theta}}(\mathbf{x}^j, \mathbf{y})$ is a sum of unary and pairwise potentials, and the graph regularizer is a high order potential. For decomposable loss functions like Hamming loss, the graph regularizer becomes a sum of submodular pairwise potentials. The MAP inference is then a standard inference problem for pairwise graphical models and can be solved via graph cuts. The structure of this graph is shown in Fig. 4.1. More complicated loss functions, such as the PASCAL loss, can also be handled using iterative dual decomposition method.

The graph regularizer can also be combined with other types of high order regularizers, for example the cardinality regularizers described earlier. In fact, graphs with submodular pairwise potentials have a known short-boundary bias [77] which favors a small number of cut edges (pairs of pixels that have different labels). This bias can cause some serious problems in SSL when the number of labeled examples is not balanced across classes. In our binary segmentation problem, usually the majority of pixels belong to background and only a small portion belong to foreground. Then when we run the optimization, this bias would make the model predict much more background for the unlabeled images. In the extreme case when unary potentials are weak, all unlabeled pixels will be predicted to have the dominant label.

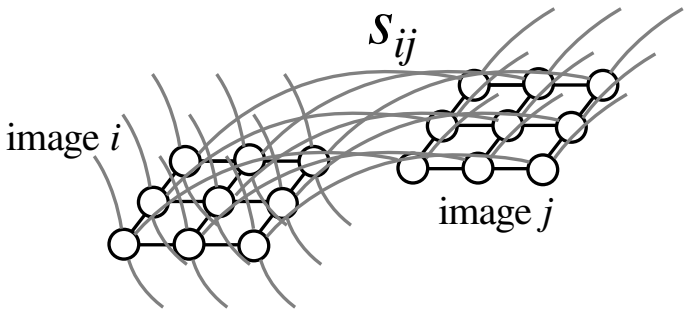


Figure 4.1: Graph structure with Hamming loss. Black edges represent intra image structure, and grey edges represent graph constraints.

The use of cardinality regularizers is then especially important.

We define a cardinality regularizer

$$R_C(\mathbf{Y}_U) = \gamma\, h(1^\top \mathbf{Y}_U) \tag{4.8}$$

where γ is a weight parameter and

$$h(x) = \max\{0, |x - x_0| - \delta\}^2 \tag{4.9}$$

x_0 is the expected number of foreground pixels computed according to the number of total pixels and the proportion of foreground in labeled images, and δ is the deviation from x_0 that can be tolerated without paying a cost. We use $\delta = x_0/5$ throughout all our experiments.

Then the optimization problem in Step 1 becomes

$$\max_{\mathbf{Y}_U} \quad \frac{\mu}{U} \sum_{j=L+1}^{L+U} F_{\boldsymbol{\theta}}(\mathbf{x}^i, \mathbf{y}^j) - \lambda \sum_{i,j:s_{ij}>0} s_{ij} \Delta(\mathbf{y}^i, \mathbf{y}^j) - \gamma\, h(1^\top \mathbf{Y}_U) \tag{4.10}$$

Finding the optimum of this problem is in general not easy. However, finding the optimum for both a submodular pairwise MRF and a cardinality potential plus unary potentials can be done very efficiently. We therefore decompose the objective into two parts and use dual decomposition [148] for the optimization. A review of dual decomposition methods is presented in Section 2.3.1, but we go over the inference details here again for completeness.

Dual Decomposition More specifically, the optimization problem in Eq. 4.10 is an instance of the following more general optimization problem

$$\max_{\mathbf{y}} \; f^u(\mathbf{y}) + f^p(\mathbf{y}) + h(\mathbf{y}) \tag{4.11}$$

where $f^u(\mathbf{y}) = \sum_i f_i^u(y_i)$ is a set of unary potentials, $f^p(\mathbf{y}) = \sum_{i,j} f_{ij}^p(y_i, y_j)$ is a set of pairwise potentials and $h(\mathbf{y})$ is a high order potential on $\mathbf{y} \in \{0,1\}^N$. To see this, note that in Eq. 4.10 $F_{\boldsymbol{\theta}}$ is a sum of unary and pairwise potentials and $\Delta(\mathbf{y}^i, \mathbf{y}^j)$ is a sum of pairwise terms for a Hamming loss.

It is usually the case that both of the two subproblems

$$\max_{\mathbf{y}} \ f^u(\mathbf{y}) + f^p(\mathbf{y}) \quad \text{and} \quad \max_{\mathbf{y}} \ f^u(\mathbf{y}) + h(\mathbf{y}) \tag{4.12}$$

are easy to solve. For example, inference for pairwise models with submodular pairwise potentials can be solved efficiently [13], and cardinality potentials with unary potentials can also be solved efficiently [51]. However, jointly optimizing the whole objective is hard.

In dual decomposition, we utilize the structure of the problem and decompose the original problem into two subproblems that are more tractable. We define $A(\mathbf{y}) = \alpha f^u(\mathbf{y}) + f^p(\mathbf{y})$ and $B(\mathbf{y}) = (1 - \alpha) f^u(\mathbf{y}) + h(\mathbf{y})$, where α is a fixed constant, e.g. 0.5, then the original objective is $A(\mathbf{y}) + B(\mathbf{y})$. Next, we can introduce auxiliary variables $\boldsymbol{\lambda} \in \mathbb{R}^N$, and upper bound the original optimization problem by

$$\mathcal{U}(\boldsymbol{\lambda}) = \max_{\mathbf{y}} \{A(\mathbf{y}) + \boldsymbol{\lambda}^\top \mathbf{y}\} + \max_{\mathbf{y}} \{B(\mathbf{y}) - \boldsymbol{\lambda}^\top \mathbf{y}\} \tag{4.13}$$

This upperbound is valid for any arbitrary $\boldsymbol{\lambda}$.

As $\boldsymbol{\lambda}^\top \mathbf{y}$ is just a sum of simple unary potentials, each of the subproblems here are easy to solve: the first subproblem has the structure of maximizing unary potentials plus pairwise potentials, and the second subproblem has the structure of maximizing unary potentials plus a high order potential.

We then minimize this upper bound over $\boldsymbol{\lambda}$, to make it as tight as possible and hence approach the optimum of the original problem. We can compute the subgradient of the upper bound with respect to $\boldsymbol{\lambda}$ as

$$\frac{\partial \mathcal{U}}{\partial \boldsymbol{\lambda}} = \hat{\mathbf{y}}^A - \hat{\mathbf{y}}^B \tag{4.14}$$

where

$$\hat{\mathbf{y}}^A = \operatorname*{argmax}_{\mathbf{y}} \ A(\mathbf{y}) + \boldsymbol{\lambda}^\top \mathbf{y} \quad \text{and} \quad \hat{\mathbf{y}}^B = \operatorname*{argmax}_{\mathbf{y}} \ B(\mathbf{y}) - \boldsymbol{\lambda}^\top \mathbf{y}. \tag{4.15}$$

In our experiments we follow this subgradient to minimize the upper bound, but a wide range of other optimization techniques can be applied here as well.

When $\hat{\mathbf{y}}^A = \hat{\mathbf{y}}^B$, it is guaranteed [148] that the $\hat{\mathbf{y}}^A$ is the optimal solution to the original problem. When this is not the case, we can choose a $\hat{\mathbf{y}}^A$ or $\hat{\mathbf{y}}^B$ encountered during the optimization process that achieves the highest score under the original objective function. Other heuristics can also be used.

4.3 Connection to Posterior Regularization

There is a surprising connection between the proposed framework and the PR based SSL method described in [57]. We show in this section that for some general settings the two methods are optimizing special cases of the same objective. The key results are: under a zero temperature limit, (1) the KL-divergence term in PR (see below) becomes the constraint violation penalty in our framework (Eq. 4.3), and (2) the posterior distribution becomes the (hard) model prediction.

The idea of PR is to regularize the posterior distributions so that they are consistent with some prior knowledge. For graph-based SSL the prior knowledge is the smoothness of the posterior distribution over the graph. PR optimizes the following objective

$$\min_{\boldsymbol{\theta}} \quad \frac{1}{L} \sum_{i=1}^{L} \ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}) + \lambda R\left(\left\{p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^j)\right\}_{j=L+1}^{L+U}\right) \tag{4.16}$$

where $\ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}) = -\log p_{\boldsymbol{\theta}}(\mathbf{y}^i|\mathbf{x}^i)$ is the negative conditional log likelihood for labeled data, and R is the posterior regularizer.

In PR, auxiliary distributions $\{q_j(\mathbf{y})\}_{j=L+1}^{L+U}$ are introduced to make the optimization easier, and the following objective is used instead:

$$\min_{\boldsymbol{\theta},q} \quad \frac{1}{L}\sum_{i=1}^{L} \ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}) + \lambda R(q) + \frac{\mu}{U}\sum_{j=L+1}^{L+U} \mathrm{KL}(q_j(\mathbf{y})||p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^j)). \tag{4.17}$$

Optimizing this objective will learn $\boldsymbol{\theta}$ and q such that the $p_{\boldsymbol{\theta}}$ distribution is consistent with labeled data, the q distribution is smooth on the graph, and the two distributions should also be close to each other in terms of KL-divergence. This objective is then optimized in an alternating approach similar to the method utilized in our model as described above.

To relate this formulation of PR to our proposed method, we introduce a temperature parameter T, and define

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}, T) = \frac{1}{Z_T^p}\exp\left(\frac{f(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})}{T}\right) \quad \text{and} \quad q(\mathbf{y}, T) = \frac{1}{Z_T^q}\exp\left(\frac{g(\mathbf{y})}{T}\right). \tag{4.18}$$

Here Z_T^p and Z_T^q are normalizing constants, and $g(\mathbf{y})$ is an arbitrary score function. The temperature augmented objective has the form of

$$\min_{\boldsymbol{\theta},q} \quad \frac{1}{L}\sum_{i=1}^{L} \ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}, T) + \lambda R(q_T) + \frac{\mu}{U}\sum_{j=L+1}^{L+U} T\mathrm{KL}(q_j(\mathbf{y}, T)||p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^j, T)) \tag{4.19}$$

where $\ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}, T) = -\log p_{\boldsymbol{\theta}}(\mathbf{y}^i|\mathbf{x}^i, T)$ and $R(q_T)$ is the regularizer defined on $\{q_j(\mathbf{y}, T)\}_{j=L+1}^{L+U}$. This objective is the same as the PR objective when $T = 1$. Next we show that when $T \to 0$ this becomes the objective of our method in Eq. 4.3.

Using the definition of p and q, the KL-divergence term can be rewritten as

$$T\mathrm{KL}(q_j(\mathbf{y}, T)||p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^j, T)) = \sum_{\mathbf{y}} q_j(\mathbf{y}, T)\left[g_j(\mathbf{y}) - f(\mathbf{x}^j, \mathbf{y}, \boldsymbol{\theta})\right] + TZ_T^p - TZ_T^q \tag{4.20}$$

Denote $\mathbf{y}^j = \mathrm{argmax}_{\mathbf{y}}\, q_j(\mathbf{y}, T)$, and let $T \to 0$, then

$$q_j(\mathbf{y}, T) \to \begin{cases} 1, & \mathbf{y} = \mathbf{y}^j \\ 0, & \text{otherwise} \end{cases} \tag{4.21}$$

and

$$TZ_T^p \to \lim_{T\to 0} T\log\sum_{\mathbf{y}}\exp\left(\frac{f(\mathbf{x}^j, \mathbf{y}, \boldsymbol{\theta})}{T}\right) = \max_{\mathbf{y}} f(\mathbf{x}^j, \mathbf{y}, \boldsymbol{\theta}) \tag{4.22}$$

$$TZ_T^q \to \lim_{T\to 0} T\log\sum_{\mathbf{y}}\exp\left(\frac{g_j(\mathbf{y})}{T}\right) = g_j(\mathbf{y}^j) \tag{4.23}$$

Substituting the above equations into Eq. 4.20,

$$T\mathrm{KL}(q_j(\mathbf{y}, T)||p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}^j, T)) \to \max_{\mathbf{y}} f(\mathbf{x}^j, \mathbf{y}, \boldsymbol{\theta}) - f(\mathbf{x}^j, \mathbf{y}^j, \boldsymbol{\theta}) \tag{4.24}$$

as $T \to 0$. This is identical to the constraint violation penalty in Eq. 4.3.

The relation between the regularizer terms depends on the specific regularizers used in the model. For example, R can be defined as $\sum_{i,j:s_{ij}>0} s_{ij} \sum_c (p_{\boldsymbol{\theta}}(y_{ic} = 1|\mathbf{x}^i) - p_{\boldsymbol{\theta}}(y_{jc} = 1|\mathbf{x}^j))^2$, where c indexes pixels, as in [57]. Here $p_{\boldsymbol{\theta}}(y_{ic} = 1|\mathbf{x}^i) = 1$ for labeled foreground pixels and $p_{\boldsymbol{\theta}}(y_{ic} = 1|\mathbf{x}^i) = 0$ for labeled background pixels, to only regularize the posterior distributions for the unlabeled data.

For the regularizer term in this case, according to Eq. 4.21, for binary segmentation we have $q_j(y_c = 1, T) \to y_{jc}$ as $T \to 0$ for each pixel c. Therefore

$$R(q_T) \to \sum_{i,j:s_{ij}>0} s_{ij} \sum_c (y_{ic} - y_{jc})^2 = \sum_{i,j:s_{ij}>0} s_{ij} \Delta(\mathbf{y}^i, \mathbf{y}^j) \tag{4.25}$$

where $\Delta(\mathbf{y}^i, \mathbf{y}^j)$ is the Hamming loss.

Finally, for $\ell(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\theta}, T)$ term, it is known, e.g., in [54], that as $T \to 0$ this term converges to the structured hinge loss.[2]

Remark. Hazan and Urtasun [54] proposed a framework that unifies the max-margin and probabilistic methods for structured prediction. Our result here can be thought of as an extension of this to semi-supervised learning of structured output problems. Moving to the max-margin formulation loses the uncertainty representation of the probabilistic models, but has the ability to specify high order constraints directly on model predictions and to use powerful discrete optimization algorithms, therefore overcoming some difficulties of inference in loopy probabilistic models. In addition, our generalized formulation also opens up the possibility of probabilistic models using temperatures other than 1, which can have some desirable properties, e.g., when T is close to 0 the posterior distribution will be much more concentrated.

4.4 Experiments

4.4.1 Datasets and Model Details

We explore the efficacy of the proposed framework on two semi-supervised foreground-background segmentation tasks. For the first task, we use the Weizmann Horse dataset [11], a fully labeled set of 328 images. For the unlabeled Horse dataset, we used images labeled "horse" in CIFAR-10 [86], which does not contain any segmented images. For the second task, we constructed a labeled set of 214 "bird" images from the PASCAL VOC 2011 segmentation data [37]. The unlabeled Bird images come from the Caltech-UCSD Bird (CUB) dataset [173]. Note that this setting of SSL is especially challenging as the unlabeled data comes from a different source than the labeled data; utilizing unlabeled examples that are extremely different than the labeled ones will hamper the performance of an SSL learning algorithm. For the unlabeled sets we therefore selected images that were similar to at least one image in the labeled set, resulting in 500 unlabeled Horse images from CIFAR-10, and 600 unlabeled Bird images from CUB. For all the images in both tasks, and their corresponding segmentations, we resize them to 32×32, which is also the size of all CIFAR-10 images.

The Bird images contain considerably more variation than the Horse images, as the birds are in a diverse set of poses and are often occluded. We found that utilizing the PASCAL birds alone for training, validation and test did not leave enough training examples to attain reasonable segmentation

[2] With a loss term added to the score function f, which can be set to 0 for $T = 1$ case for exact equivalence.

performance. We thus created an additional labeled set of 600 bird images using the CUB dataset (a different set of 600 images than the aforementioned unlabeled set). The CUB dataset contains rough bird segmentations of low quality. We used a grab-cut [137] like method to refine the rough segmentations and the refined segmentations are used as ground truth.

In our experiments we compare four types of models: (1) the baseline **Initial** model, which forms the basis for each of the others, and is a model trained with standard supervised learning; (2) a **Self-Training** model that iteratively uses the current model to predict labels for all unlabeled data[3] and updates itself using these predictions as true labels; (3) **Graph**, our graph-based SSL method that uses the graph regularizer R_G; (4) **Graph-Card**, our SSL method utilizing both graph and cardinality regularizer $R_G + R_C$.

The Initial model is trained in a fully supervised way on only labeled data by subgradient decent on scaled structured hinge loss. The model's score function F is a model with unary and pairwise terms. We extracted a 149 dimensional descriptor for each pixel in an image by applying a filter bank. Then a multi-layer neural network is trained using these descriptors as input to predict binary labels[4]. The log probability of each class is used as the unary potential. For pairwise potentials, we used a standard 4-connected grid neighborhood and the common Potts model, where $f^p(y_i, y_j, \mathbf{x}) = -p_{ij}\mathbf{I}[y_i \neq y_j]$ and p_{ij} is a penalty for assigning different labels for neighboring pixels y_i and y_j. We define p_{ij} as the sum of a constant term that encourages smoothing and a local contrast sensitive term defined in [14] which scales down the penalty when the RGB difference between pairs of pixels is large. In our experiments, we fix the pairwise potentials and focus on learning parameters in the neural network for unary potentials only, which contains the vast majority of the parameters.

During learning, the gradients are back-propagated through the neural network to update parameters. Since neural networks are highly nonlinear models, it is hard to find the optimal $\boldsymbol{\theta}$ in Eq. 4.5 in every Step 2 of our algorithm. Instead, we only take a few gradient steps in Step 2 of each iteration. Other hyper parameters, e.g. λ, μ, γ, are tuned using the validation set.

For the graph-based models, we used the Histogram of Oriented Gradients (HOG) [27] image features to construct the graph. We set $s_{ij} = 1$ if examples i and j are one of each other's 5 nearest neighbors based on the Euclidean distance between the HOG features, and $s_{ij} = 0$ otherwise. Fig. 4.2 shows some nearest neighbor search results using HOG distance.

4.4.2 Experimental Settings

For our experiments, we examine how the performance of the SSL methods change with the number of labeled images, by randomly selecting L images from the training set to be used as labeled data and adding the remaining images to the unlabeled set. Starting from $L = 5$, we gradually increase L to the entire training set. Note that while we vary the training and unlabeled sets in this way, the validation and test sets remain constant, in order to make comparisons fair. This process is repeated 10 times, each time including randomly selected images in the training set. All models are evaluated using per-pixel prediction accuracy averaged over pixels in all images, and we report the mean and standard deviation of the results over the 10 repetitions.

We ran three types of experiments. In the first one, the training, validation and test set were all drawn from the same dataset. For the Horse task, there were up to 200 training images, 48 validation,

[3] We didn't choose the most confident model predictions, as estimating confidence is nontrivial for structured predictions.
[4] We also tried a linear model initially, but neural nets significantly outperform linear models by about 10%.

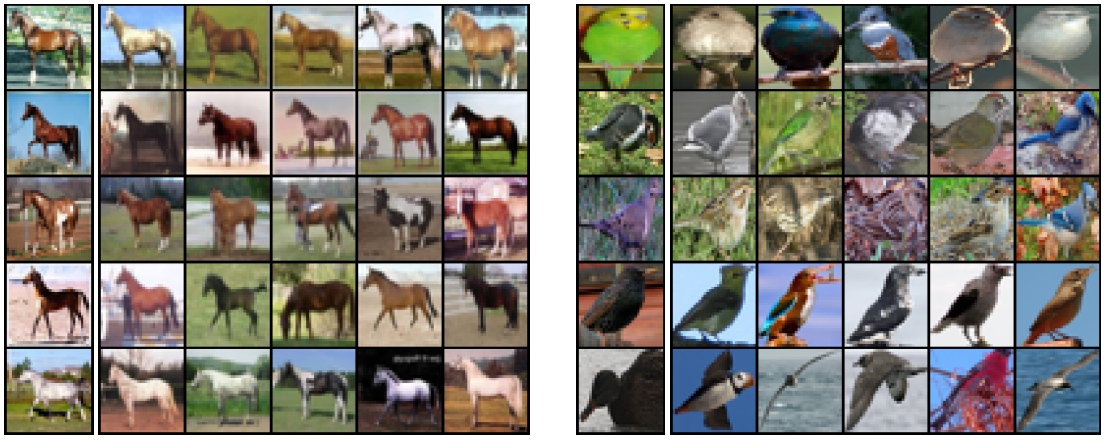


Figure 4.2: Left most column are query images, and the 5 columns on the right are the nearest neighbors retrieved based on HOG similarity. All query images are randomly chosen. Left: query from Weizmann dataset, retrieve CIFAR-10 horses. Right: query from PASCAL dataset, retrieve CUB birds.

and 80 test images, drawn from the Weizmann set, and 500 unlabeled images from CIFAR-10. For the Bird task, there were up to 200 training images, 200 validation, and 200 test images, and 600 unlabeled images, all drawn from the CUB dataset.

The second experiment explored domain adaptation. In many experimental settings, there are insufficient labeled examples to obtain good performance. This was the case with our PASCAL Bird dataset, which necessitated labeling examples from the CUB set. An interesting question is whether training on one domain, the source domain, can transfer to a different, target domain, when the unlabeled data comes from the target domain, i.e., the same dataset as the test set, and both differ from the training set. It is possible for the model to learn special features about the target domain by using unlabeled data, therefore obtaining larger performance gains. In the second experiment we explored the performance of the various models in a version of this domain adaptation setting on the Bird segmentation task.

The third experiment directly assesses the impact of drawing the validation set from the same dataset as the source, versus drawing the validation from the target domain. In our original bird experiment the validation set comes from the source domain, while in the second experiment it comes from the target domain; tuning hyperparameters on the target domain may contribute to some of the performance gains. To examine this, we compared the models in two more settings, both of which use a training set of 40 images drawn from the PASCAL dataset, and the same 200 CUB test images and 600 unlabeled CUB images. The experiments differ in that in the first setup the validation set is composed of 174 images drawn from the source domain, the PASCAL set, while in the second they are from the target CUB domain. Table 4.1 lists the datasets used in each experimental setting.

4.4.3 Results

Experiment 1. Results for the first basic SSL experiments are shown in Fig. 4.3; (a),(c) show how test set performance changes as the number of labeled images increases, while Fig. 4.3(b),(d) show the improvement over the initial model from the three different SSL methods.

As can be seen, for both segmentation tasks self-training achieves a small improvement with very few labeled examples, but does not help too much in general, as it is mostly reinforcing the model itself.

Experiment	train	validation	test	unlabeled
(1) Horse	W-200$^-$	W-48	W-80	R-500$^+$
(1) Bird	C-200$^-$	C-200	C-200	C-600$^+$
(2) Domain Adapt.	P-214$^-$	C-200	C-200	C-600$^+$
(3) Val: Source	P-40$^-$	P-174	C-200	C-600$^+$
(3) Val: Target	P-40$^-$	C-174	C-200	C-600$^+$

Table 4.1: Experimental settings and datasets. Each dataset description follows the format [dataset code]-[size]. Dataset codes: P for PASCAL VOC birds, C for CUB birds, W for Weizmann horses, R for CIFAR-10 horses. Superscript "-" means at most, and "+" means at least, see paper for more details.

Graph-based methods work significantly better than self-training throughout. For Horse segmentation, the use of unlabeled data helps the most when the number of labeled images is small. The improvement becomes smaller as the number of images increases. The model saturates and achieves very high accuracy (more than 92%) with 200 labeled images, where using unlabeled data does not make too much difference.

For Bird segmentation, graph-based methods achieve a small improvement over self-training and the initial model when the number of labeled images is small ($L \leq 20$). This can be explained by the complexity of the bird dataset; more examples are required to achieve reasonable segmentations. There is a jump in performance from $L = 20$ to $L = 40$: as the initial model gets better, combining with the graph, inferred labels for unlabeled data become much better and therefore more helpful. From Fig. 4.3 we can see that when $L = 40$, using graph-based methods the test accuracy nearly matches that of a fully supervised model trained with all 200 labeled images, thus saving a lot of labeling work.

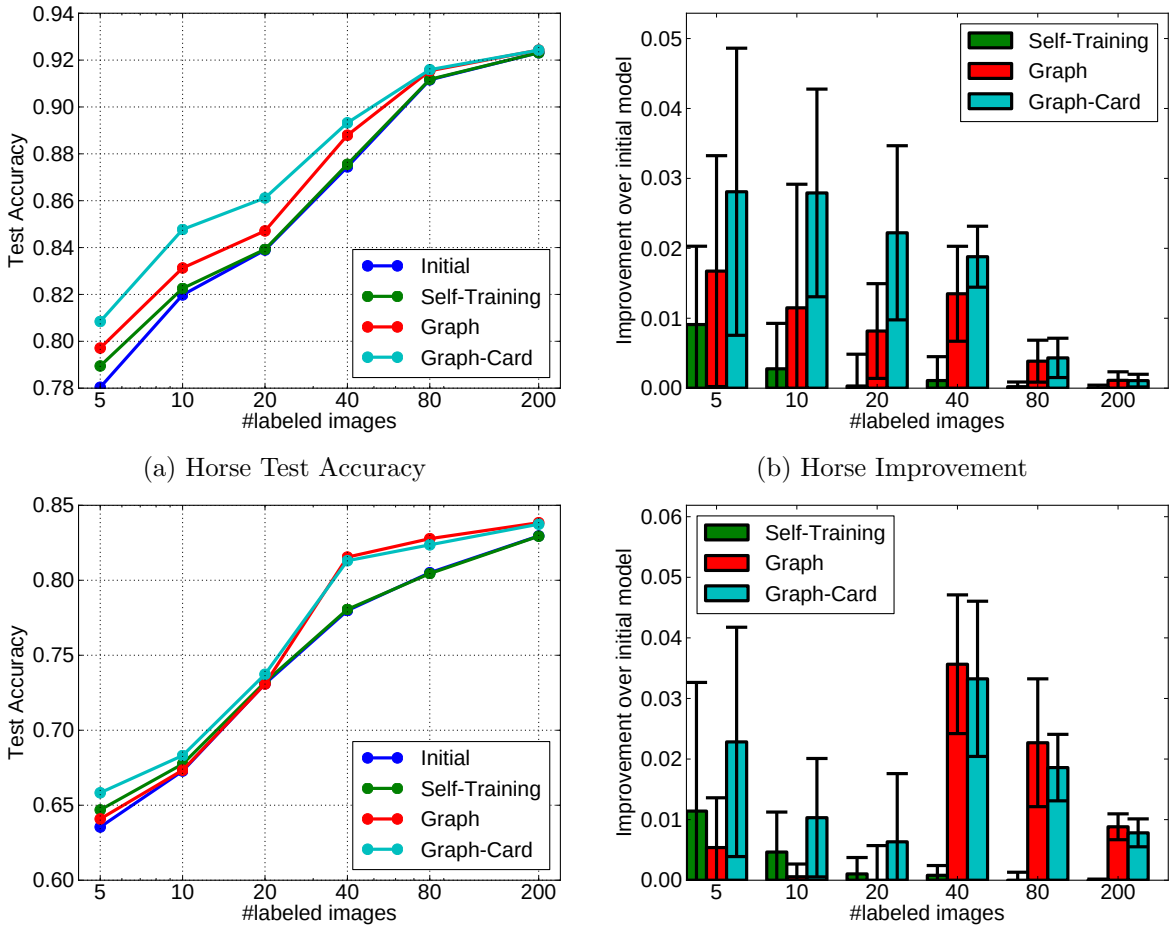
Comparing "Graph-Card" and "Graph", we can see that using a cardinality regularizer further improves performance over only using the graph regularizer, as in most horse segmentation cases and bird segmentation with few labeled images. It is most helpful when the number of images is small, where the initial model is very weak and the short-boundary bias becomes especially significant when inferring labels for unlabeled images. For a lot of cases, the use of cardinality potentials can compensate for this bias.

Experiment 2. Fig. 4.4 shows the results for the domain adaptation setting, where the training data is from one dataset while the unlabeled data and the test and validation examples come from a different set. Compared to the original bird experiment, we observe that: (1) the performance jump from $L = 20$ to $L = 40$ is considerably larger; (2) the gap between SSL methods and the initial model is also more significant; and (3) the improvement from self-training is almost non-existent.

Experiment 3. We compare the "Graph-Card" method across the two settings, where the validation set is either from the source or the target domain. Fig. 4.5 summarizes the results. In this comparison, the model validated on the target domain performs consistently better than the model validated on the source domain. However, the difference decreases as the number of labeled images increases, as in both settings the method is getting closer to the limit, which can be seen from other experiments on bird segmentation, where the performance levels off when $L \geq 40$.

4.5 Discussion

In this chapter, we proposed a new framework for semi-supervised structured output learning that allows the use of expressive high order regularizers defined directly on model predictions for unlabeled data. We proved that this framework and PR are closely related. Experimental results on image segmentation



(a) Horse Test Accuracy

(b) Horse Improvement

(c) Bird Test Accuracy

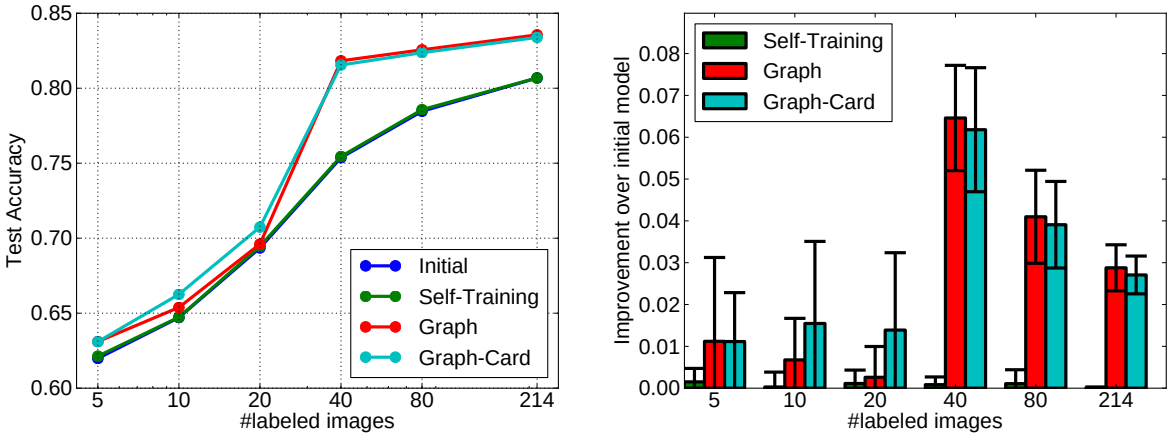
(d) Bird Improvement

Figure 4.3: Experiment 1 (a),(c): Test performance for the initial model and the 3 SSL methods; (b),(d): improvements for the three methods over the initial model.

tasks demonstrated the effectiveness of our framework, and its ability to strongly benefit from unlabeled data in a domain adaptation setting.

Looking forward, exploring the learning of the input similarity metric s_{ij} in our graph-based SSL, and also incorporating other types of high order regularizers are promising directions for research. Developing more efficient inference algorithms for these high order regularizers is important for the success of the method. On the application side, our segmentation tasks are especially relevant when combined with an object detector. SSL for a structured prediction model that performs segmentation and detection jointly is an interesting and challenging future direction.

From our empirical experience, the most important key to the success of these semi-supervised learning methods is to have a good similarity metric between examples. Examples that are close under such a similarity metric should have similar output labels. For classification problems, if we have access to a perfect similarity metric which puts examples having the same class close to each other and examples in different classes far apart, then we effectively have access to fully labeled data. For structured output problems, it is very rare for two examples to have exactly the same label, and even knowing the perfect similarity metric does not imply we know the true label for unlabeled data. This implies that semi-



(a) Acc vs. #labeled (b) Improvement

Figure 4.4: Experiment 2: Results for the domain adaptation Bird task, where the unlabeled and validation and test sets are from a different dataset than the training set. The curve for "Initial" is behind "Self-Training".

supervised learning for structured output problems are a lot harder than for simple prediction problems. However, combined with a prediction model, a better similarity metric in general leads to better semi-supervised learning performance.

In our experiments, the HOG based feature representations for images provided a good similarity metric, as HOG captures object shape quite well, and objects with similar shapes, meaning similar sizes, orientations, etc., have similar segmentation labels. More generally, similarity metric learning may be framed as a representation learning problem, which maps data examples with similar labels to similar representations. This perspective makes the development of neural network based representation learning methods very relevant.

On the other hand, if a similarity metric is not great, the success of semi-supervised learning requires the similarity metric and the prediction model to capture different aspects of the prediction. This makes it possible for the model to learn from the similarity metric. It also requires that we start with a reasonably good model, because if the starting model is bad, then a far-from-perfect similarity metric may lead it to become even worse.

From a more practical side, semi-supervised learning is a way to utilize unlabeled data to improve a supervised learning system. Other techniques may be also relevant in terms achieving this goal. In the case of having only a very small set of labeled data, an alternative to directly applying semi-supervised learning is to simply make an effort to label more data. In this case, if it is difficult to train a reasonably good model with supervised learning, then it is not guaranteed that semi-supervised learning may help that much. The labeling effort can be optimized by using techniques like active learning.

The setup where semi-supervised learning is really attractive is when the amount of training data is enough to train a reasonably good model, but we also have magnitudes more unlabeled data that can be used to further improve the model. At this scale, hand labeling meaningfully more data is not feasible, therefore good semi-supervised learning algorithms should really excel in this setup.

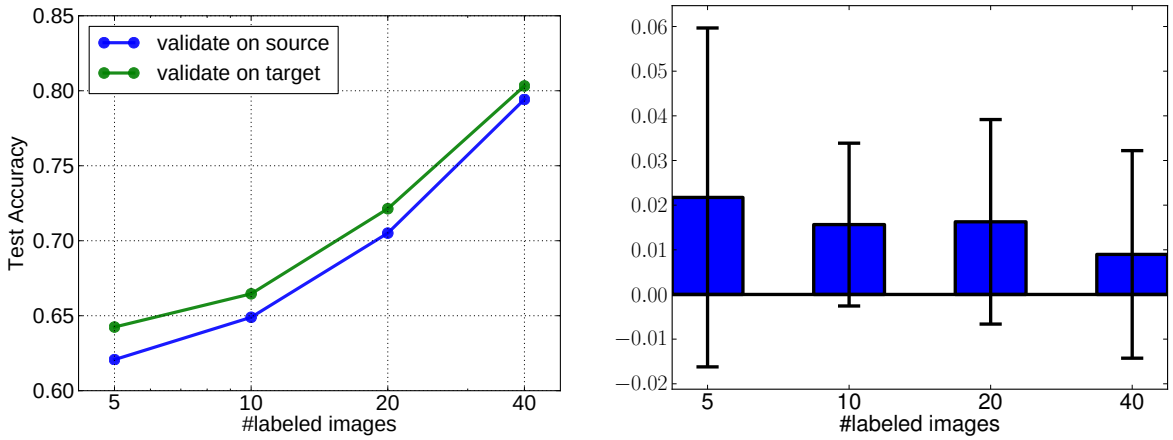


Figure 4.5: Experiment 3: Comparison between validation on source domain and validation on target domain. Left: test accuracy as number of labeled images increases. Right: difference between the two settings (validate on target vs. validate on source).

Chapter 5

Neural Mean Field Networks

In this chapter, we take a closer look at the relationship between graphical models, commonly used for representation and reasoning of structured data and problems, with neural networks. In the study, we found that these two types of models have an interesting connection. More specifically, in this chapter we show that pairwise Markov random field (MRF) models, including its conditional variants like CRFs, when used for prediction through an iterative mean field inference algorithm, can be converted equivalently into a neural network with a special structure and weight tying scheme.

The theoretical result in this chapter is generally applicable to pairwise MRF models with mean field inference, and can be extended to other iterative inference algorithms like belief propagation and its variants, as well as MRFs with high order potentials. The equivalence result connects a quite general class of MRFs to a constrained type of neural network. Realizing this motivates us to relax the constraints on the neural network models, which can further increase the model capacity and has shown promise in preliminary experiments.

The results obtained in this chapter also indicate that neural network models may well be used as stand-alone structured prediction models.

5.1 Equivalence Between Mean Field Inference for Pairwise MRFs and Neural Networks

We consider pairwise MRFs defined for random vector $\mathbf{y}$ on graph $G = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$ of the following form,

$$p(\mathbf{y}) = \frac{1}{Z} \exp(F_{\boldsymbol{\theta}}(\mathbf{y})), \tag{5.1}$$

where the scoring function $F_{\boldsymbol{\theta}}(\mathbf{y})$ is a sum of unary (f_i) and pairwise (f_{ij}) potentials

$$F_{\boldsymbol{\theta}}(\mathbf{y}) = \sum_{i \in \mathcal{V}} f_i(y_i, \boldsymbol{\theta}) + \sum_{(i,j) \in \mathcal{E}} f_{ij}(y_i, y_j, \boldsymbol{\theta}) \tag{5.2}$$

$\boldsymbol{\theta}$ is a set of parameters in F and $Z = \sum_{\mathbf{y}} \exp(F_{\boldsymbol{\theta}}(\mathbf{y}))$ is a normalizing constant. We assume for all $i \in \mathcal{V}$, y_i takes values from a discrete set $\{1, ..., K\}$. Note that $p(\mathbf{y})$ can be a posterior distribution $p(\mathbf{y}|\mathbf{x})$ (a CRF) conditioned on some input $\mathbf{x}$, and the scoring function can depend on $\mathbf{x}$ with parameter $\boldsymbol{\theta}$. We do not make this dependency explicit for simplicity of notation, but all discussions in this chapter apply to

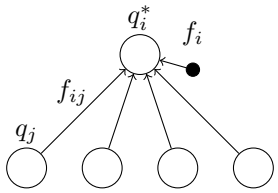


Figure 5.1: Illustration of one unit in Mean Field Networks.

conditional distributions just as well and most of our applications are for conditional models. Pairwise MRFs are widely used in, for example, image segmentation, denoising, depth estimation, etc. Inference in such models is hard in general.

The mean field algorithm is a widely used approximate inference algorithm, and is described in Section 2.3.2. The algorithm finds the best factorial distribution $q(\mathbf{y}) = \prod_{i \in \mathcal{V}} q_i(y_i)$ that minimizes the KL-divergence with the original distribution $p(\mathbf{y})$. The standard strategy to minimize this KL-divergence is coordinate descent. When fixing all q_j distributions except q_i, the optimal distribution $q_i^*(y_i)$ has a closed form solution

$$q_i^*(y_i) = \frac{1}{Z_i} \exp\left(f_i(y_i, \boldsymbol{\theta}) + \sum_{j \in \mathcal{N}(i)} \sum_{y_j} q_j(y_j) f_{ij}(y_i, y_j, \boldsymbol{\theta}) \right) \tag{5.3}$$

where $\mathcal{N}(i)$ represents the neighborhood of vertex i and Z_i is a normalizing constant. In each iteration of mean field, the q distributions for all variables are updated in turn and the algorithm is executed until some convergence criterion is met.

We observe that Eq. 5.3 can be interpreted as a feed-forward operation similar to those used in neural networks. More specifically, q_i^* corresponds to the output of a node and q_j's are the outputs of the layer below, f_i are biases and f_{ij} are weights, and the nonlinearity for this node is a softmax function. Fig. 5.1 illustrates this correspondence. Note that unlike ordinary neural networks, the q nodes and biases are all vectors, and the connection weights are matrices.

Based on this observation, we can map a M-iteration mean field algorithm to a M-layer feed-forward neural network. Each iteration corresponds to the forward mapping from one layer to the next, and all layers share the same set of weights and biases given by the underlying graphical model. The bottom layer contains the initial distributions. We call this type of network a Neural Mean Field Network (NMFN).

Fig. 5.2 shows 2-layer NMFNs for a chain of 4 variables with different update schedule in mean field inference. Though it is possible to do exact inference for chain models, we use them here just for illustration. Note that the update schedule determines the structure of the corresponding NMFN. Fig. 5.2(a) corresponds to a sequential update schedule and Fig. 5.2(b) corresponds to a block parallel update schedule.

From the feed-forward neural network point of view, NMFNs are just a special type of feed-forward neural network, with a few important restrictions on the network architecture:

- The weights and biases, or equivalently the parameters $\boldsymbol{\theta}$, on all layers are tied and equal to the $\boldsymbol{\theta}$ in the underlying pairwise MRF.

- The network structure is the same on all layers and follows the structure of the pairwise MRF.

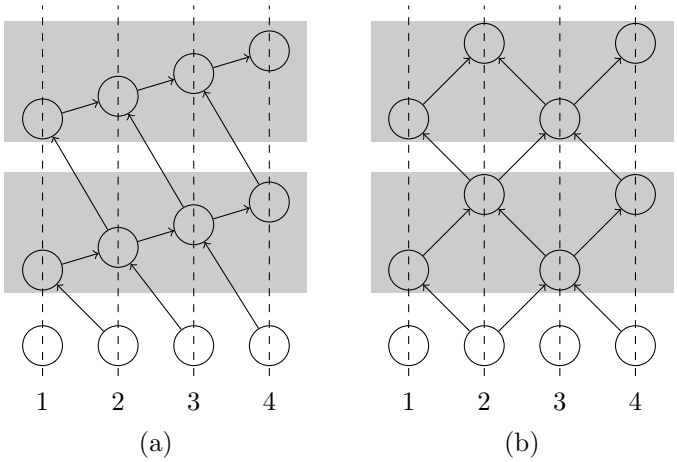


Figure 5.2: 2-layer NMFNs for a chain model ①-②-③-④ with (a) sequential update schedule, (b) block parallel update schedule. The weights and biases are dropped. The grey plates indicate layers. The height of a node indicates its order in the updates.

These two restrictions make M-layer NMFNs exactly equivalent to M iterations of the mean field algorithm. But from the feed-forward neural network point of view, nothing stops us from relaxing the restrictions, as long as we keep the number of outputs at the top layer unchanged, making sure we are predicting the right target.

5.2 Relaxing the Restrictions on the NMFNs

By relaxing the restrictions discussed above, we lose the equivalence to mean field, but if all we care about is the quality of the input-to-output mapping, measured by some loss function like KL-divergence, then this relaxation can be beneficial. We discuss a few relaxations here that aim to improve M-layer NMFNs with fixed M as an inference tool for a pairwise MRF with fixed θ:

(1) Untying $\boldsymbol{\theta}$'s in NMFNs from the $\boldsymbol{\theta}$ in the original pairwise MRF. If we consider M-layer NMFNs with fixed M, then this relaxation can be beneficial as the mean field algorithm is designed to run until convergence, but not for a specific M. Therefore choosing some $\boldsymbol{\theta}' \neq \boldsymbol{\theta}$ may lead to better KL-divergence in M steps when M is small. This can save time as the same quality outputs are obtained with fewer steps. As M grows, we expect the optimal $\boldsymbol{\theta}'$ to approach $\boldsymbol{\theta}$.

(2) Untying $\boldsymbol{\theta}$'s on all layers, i.e. allow different $\boldsymbol{\theta}$'s on different layers. This will create a strictly more powerful model with many more parameters. The $\boldsymbol{\theta}$'s on different layers can therefore focus on different things; for example, the lower layers can focus on getting to a good area quickly and the higher layers can focus on converging to an optimum fast.

(3) Untying the network structure from the underlying graphical model. If we remove connections from the NMFNs, the forward pass in the network can be faster. If we add connections, we create a strictly more powerful model. Information flows faster on networks with long range connections, which is usually helpful. We can further untie the network structure on all layers, i.e. allow different layers to have different connection structures. This creates a strictly more flexible model.

Using NMFNs to Improve Inference As an example, we consider relaxation (1) for a trained pairwise CRF with parameter $\boldsymbol{\theta}$. As the model is conditioned on input data, the potentials will be different for each data case, but the same parameter $\boldsymbol{\theta}$ is used to compute the potentials. The aim here is to use a different set of parameters $\boldsymbol{\theta}'$ in NMFNs to speed up inference for the CRF with parameter $\boldsymbol{\theta}$ at test time, or equivalently to obtain better outputs within a fixed inference budget. To get $\boldsymbol{\theta}'$, we compute the potentials for all data cases first using $\boldsymbol{\theta}$. Then the distributions defined by these potentials are used as targets, and we train our NMFN to minimize the KL-divergence between the approximation and the targets. Using KL-divergence as the loss function, this training can be done by following the gradients of $\boldsymbol{\theta}'$, which can be computed by the standard back-propagation algorithm developed for feed-forward networks. To be more specific, the KL-divergence loss is defined as

$$\mathrm{KL}(q^M||p) = \sum_{i\in\mathcal{V}}\sum_{y_i\in\mathcal{Y}} q_i^M(y_i)\log q_i^M(y_i) - \sum_{i\in\mathcal{V}}\sum_{y_i\in\mathcal{Y}} q_i^M(y_i)f_i(y_i) - \sum_{(i,j)\in\mathcal{E}}\sum_{y_i,y_j\in\mathcal{Y}} q_i^M(y_i)q_j^M(y_j)f_{ij}(y_i,y_j) + C \tag{5.4}$$

where q^M is the Mth layer output of the NMFN and C is a constant representing terms that do not depend on q^M. In this objective, the potentials f_i and f_{ij} are computed with the CRF parameters $\boldsymbol{\theta}$, and the approximate marginals q^M are computed with NMFN parameters $\boldsymbol{\theta}'$. The gradient of the loss with respect to $q_i^M(y_i)$ can be computed as

$$\frac{\partial \mathrm{KL}}{\partial q_i^M(y_i)} = \log q_i^M(y_i) + 1 - f_i(y_i) - \sum_{j\in\mathcal{N}(i)}\sum_{y_j\in\mathcal{Y}} q_j^M(y_j)f_{ij}(y_i,y_j) \tag{5.5}$$

The gradient with respect to $\boldsymbol{\theta}'$ follows from the chain rule, as q^M is a function of $\boldsymbol{\theta}'$.

At test time, $\boldsymbol{\theta}'$ instead of $\boldsymbol{\theta}$ is used to compute the outputs through the NMFN, which is expected to get to the same results as using mean field in fewer steps, because the model is explicitly trained to do so.

Other loss functions, for example $\mathrm{KL}(p\|q^M)$ as used in expectation propagation [121] can also be used to train the NMFN inference network.

Using NMFNs as Stand-Alone Prediction Models The discussions above focus on making NMFNs better tools for inference. We can, however, take a step even further, to abandon the underlying pairwise MRF and use NMFNs directly as discriminative models. For this setting, NMFNs correspond to conditional distributions of form $q_{\boldsymbol{\theta}'}(\mathbf{y}|\mathbf{x})$ where $\mathbf{x}$ is some input and $\boldsymbol{\theta}'$ is the parameters. The q distribution is factorial, and defined by a forward pass of the network. The weights and biases on all layers as well as the initial distribution at the bottom layer can depend on $\mathbf{x}$ via functions with parameters $\boldsymbol{\theta}'$. These discriminative NMFNs can be learned using a training set of $(\hat{\mathbf{x}},\hat{\mathbf{y}})$ pairs to minimize some loss function. An example is the element-wise hinge loss, which is better defined on the output layer activations

$$a_i^*(y_i) = f_i(y_i) + \sum_{j\in\mathcal{N}(i)}\sum_{y_j} q_j(y_j)f_{ij}(y_i,y_j), \tag{5.6}$$

which is simply the exponent part in Eq. 5.3. The loss is

$$\ell(\mathbf{a}^M,\hat{\mathbf{y}}) = \sum_{i\in\mathcal{V}}\left[\max_k\left\{a_i^M(k)+\Delta(k,\hat{y}_i)\right\} - a_i^M(\hat{y}_i)\right] \tag{5.7}$$

Figure 5.3: Three pairs of example images, in each pair: left image is the noisy input image, right image is the ground truth label.

where Δ is the task loss function. An example is $\Delta(k, \hat{y}_i) = c\mathbf{I}[k \neq \hat{y}_i]$, where c is the loss for mislabeling and $\mathbf{I}[.]$ is the indicator function. The gradient of this loss with respect to a^M has a very simple form

$$\frac{\partial \ell}{\partial a_i^M(k)} = \mathbf{I}[k = k^*] - \mathbf{I}[k = \hat{y}_i] \tag{5.8}$$

where $k^* = \text{argmax}_k \left\{a_i^M(k) + \Delta(k, \hat{y}_i)\right\}$ is the model prediction. The gradient of $\boldsymbol{\theta}'$ can then be computed using back-propagation.

Compared to the standard paradigm that uses intractable inference during learning, these discriminative NMFNs are trained with fixed inference budget (M steps/layers) in mind, and therefore can be expected to work better when we only run the inference for a fixed number of steps. The discriminative formulation also enables the use of a variety of different loss functions more suitable for discriminative tasks like the hinge loss defined above, which is usually not straight-forward to be integrated into the standard paradigm. Many relaxations described before can be used here to make the discriminative model more powerful, for example untying weights on different layers.

5.3 Preliminary Experiment Results

We demonstrate the performance of NMFNs on an image denoising task. We generated a synthetic dataset of 50×100 images. Each image has a black background (intensity 0) and some random white (intensity 1) English letters as foreground. Then independent flipping noise (pixel intensity flipped from 0 to 1 or 1 to 0) and Gaussian noise are added to each pixel. The task is to recover the clean text images from the noisy images, more specifically, to label each pixel into one of two classes: foreground or background. This problem has the same structure as a binary segmentation problem. We generated training and test sets, each containing 50 images. A few example images and corresponding labels are shown in Fig. 5.3.

The baseline model we consider in the experiments is a pairwise CRF. The model defines a posterior distribution of output label $\mathbf{y}$ given input image $\mathbf{x}$. For each pixel i the label $y_i \in \{0, 1\}$. The conditional unary potentials are defined using a linear model $f_i(y_i, \mathbf{x}) = y_i \mathbf{w}^\top \phi(\mathbf{x}, i)$, where $\phi(\mathbf{x}, i)$ extracts a 5×5 window around pixel i and padded with a constant 1 to form a 26-dimensional feature vector, $\mathbf{w} \in \mathbb{R}^{26}$ is the parameter vector for unary potentials. The pairwise potentials are defined as Potts potentials, $f_{ij}(y_i, y_j, \mathbf{x}) = -p_{ij}\mathbf{I}[y_i = y_j]$, where p_{ij} is the penalty for pixel i and j to take different labels. We use one single penalty p_h for all horizontal edges and another p_v for all vertical edges. In total, the baseline model specified by $\boldsymbol{\theta} = (\mathbf{w}, p_h, p_v)$ has 28 parameters.

For all inference procedures in the experiments for both mean field and NMFNs, the distributions are initialized by taking softmax of unary potentials.

We learn $\boldsymbol{\theta}$ for the baseline model by gradient ascent to maximize the conditional log likelihood of training data. To compute the gradients, the posterior expectations are approximated using marginals

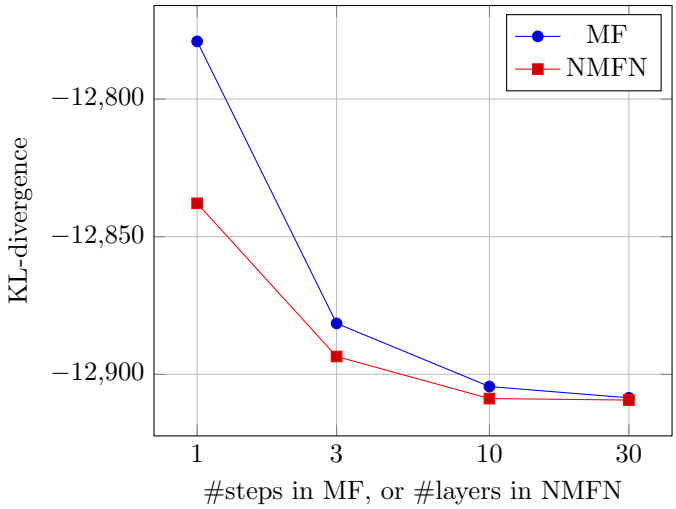


Figure 5.4: Results for the inference experiment, comparing the KL-divergence between inferred approximate distribution $q(\mathbf{y})$ and the model distribution $p_{\boldsymbol{\theta}_{\text{MF}}}(\mathbf{y}|\mathbf{x})$, where the approximate q's is obtained with M-step mean field and M-layer NMFN with different M's. Note the log partition function of $p_{\boldsymbol{\theta}_{\text{MF}}}(\mathbf{y}|\mathbf{x})$ is not included in the KL-divergence computation, as it is hard to compute and it is a irrelevant constant for the purpose of comparing different approximations.

obtained by running mean field for 30 steps (abbreviated as MF-30). $\boldsymbol{\theta}$ is initialized as an all 1 vector, except that the weight for constant feature in unary model is set to $-5 \times 5/2 = -12.5$ corresponding to roughly the mean pixel intensity. We denote this initial parameter setting as $\boldsymbol{\theta}_0$, and the parameters after training as $\boldsymbol{\theta}_{\text{MF}}$. With MF-30, $\boldsymbol{\theta}_0$ achieves an accuracy of 0.7957 on test set, which means the initialization is already pretty good; after training, the accuracy improves to 0.8109.

5.3.1 NMFN for Inference

In the first experiment, we learn NMFNs to do inference for the CRF model with fixed parameter $\boldsymbol{\theta}_{\text{MF}}$. In this task, NMFNs are used to approximate the posterior $p_{\boldsymbol{\theta}_{\text{MF}}}(\mathbf{y}|\mathbf{x})$ as well as possible. We train M-layer NMFNs (NMFN-M) with fully untied weights on all layers to minimize the KL-divergence loss for $M = 1, 3, 10, 30$. The NMFN parameters on all layers are initialized to be the same as θ_{MF}.

Results of this experiment are shown in Fig. 5.4. As baselines, the average KL-divergence on test set using mean field inference MF-1, MF-3, MF-10 and MF-30 are -12779.05, -12881.50, -12904.43, -12908.54. Note that these numbers are the KL-divergence without the constant corresponding to log-partition function, which we cannot compute and is an irrelevant constant for the purpose of comparing different approximate distributions. This explains why the KL-divergence values are negative. The corresponding KL-divergence on test set for NMFN-1, NMFN-3, NMFN-10, NMFN-30 are -12837.87, -12893.52, -12908.80, -12909.34. We can see that NMFNs improve performance more significantly when M is small, and NMFN-10 is even better than MF-30, while MF-30 runs the inference for 20 more iterations than NMFN-10.

The inferred marginals for the mean field baselines and the learned NMFN-1 model on an example image are shown in Fig. 5.5 (zoom in for details). For visualization we show the marginal probability for foreground. We can see the qualitative differences of results given by different methods, where the results of NMFN-1 is considerably cleaner than the results of MF-1.

G.T. MF-1, 3, 10, 30 NMFN-1

Figure 5.5: Inference example for an image, comparing inference results with mean field inference versus results obtained with NMFN-1. G.T.: ground truth.

Model	CRF + Mean Field			NMFN	
#steps or #layers	30		3	3	
Parameters	$\boldsymbol{\theta}_0$	$\boldsymbol{\theta}_{\text{MF}}$	$\boldsymbol{\theta}_{\text{MF}}$	tied	untied
Accuracy	0.7957	0.8109	0.8065	0.8134	0.8151

Table 5.1: Results for NMFNs as discriminative models. Comparing NMFN-3 variants with CRF models using mean field inference. The results are pixel accuracy on the test set.

5.3.2 NMFN as Discriminative Model

In the second experiment, we train NMFNs as discriminative models for the denoising task directly. The denoising results are shown in Table 5.1. We start with a three-layer NMFN with tied weights. The NMFN parameters are initialized to be the same as $\boldsymbol{\theta}_{\text{MF}}$. As baselines, MF-3 with $\boldsymbol{\theta}_{\text{MF}}$ achieves an accuracy of 0.8065 on test set, and MF-30 with $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_{\text{MF}}$ achieves accuracy 0.7957 and 0.8109 respectively as mentioned before.

We learn the NMFN parameters to minimize the element-wise hinge loss with learning rate 0.0005 and momentum 0.5 as discussed in Section 5.2. After 50 gradient steps, the test accuracy improves and converges to around 0.8134, which beats all the mean field baselines and is even better than MF-30 with $\boldsymbol{\theta}_{\text{MF}}$.

Then we untie the weights of the three-layer NMFN and continue training with larger learning rate 0.002 and momentum 0.9 for another 200 steps. The test accuracy improves further to around 0.8151. During learning, we observe that the gradients for the three layers are usually quite different: the first and third layer gradients are usually much larger than the second layer gradients. This may cause a problem for NMFN with tied weights, which is essentially using the same gradient (sum of gradients on three layers) for all three layers.

As a comparison, we tried to continue training NMFN with tied weights using learning rate 0.002 and momentum 0.9. The test accuracy improves to around 0.8145 but oscillated a lot and eventually diverged. We've tried a few smaller learning rate and momentum settings but can not get the same level of performance as NMFN with untied weights within 200 steps.

5.4 Extention to Loopy Belief Propagation

The idea of NMFN can be easily generalized to other message passing based iterative inference algorithms. Belief propagation methods are a natural next step.

In Section 2.3.2, the update equations for loopy belief propagation (LBP) are presented in Equations 2.27, 2.28, 2.29, 2.30. The message passing process in LBP can be unrolled in a similar way to how we unrolled the mean field inference. Consider again a pairwise model for simplicity, the messages are

computed as

$$m_{i\to ij}(y_i) = \prod_{j':(i,j')\in\mathcal{E},j'\neq j} \exp(f_i(y_i))m_{ij'\to i}(y_i) \tag{5.9}$$

$$m_{ij\to i}(y_i) = \sum_{y_j} \exp(f_{ij}(y_i,y_j))m_{j\to ij}(y_j), \tag{5.10}$$

where ij is an unordered pair. Based on these two equations, we can introduce two types of nodes $m_{i\to ij}$ and $m_{ij\to i}$ into a feed-forward network, which results in 4 network nodes per edge in one round of updates. It is usually a good idea to normalize these messages in each update, and to do computations in the log-domain to avoid numeric problems.

After this message passing process is unrolled for T steps, we can apply the output model and compute

$$q(y_i,y_j) \propto \exp(f_{ij}(y_i,y_j))m_{i\to ij}(y_i)m_{j\to ij}(y_j) \tag{5.11}$$

$$q(y_i) \propto \exp(f_i(y_i)) \prod_{j:(i,j)\in\mathcal{E}} m_{ij\to i}(y_i) \tag{5.12}$$

Once we got these approximate marginals, we can follow the discussion for NFMNs to (1) relax the restrictions; (2) use these networks as inference tools and (3) use these networks as discriminative models directly. The extentions to other message passing based algorithms can be derived by following this process easily.

5.5 Related Work

Previous work by Justin Domke [32, 33] and Stoyanov et al.[150] are the most related to ours. In [32, 33], the author described the idea of truncating message passing at learning and test time to a fixed number of steps, and back-propagating through the truncated inference procedure to update parameters of the underlying graphical model. In [150] the authors proposed to train graphical models in a discriminative fashion to directly minimize empirical risk, and used back-propagation to optimize the graphical model parameters.

Compared to their approaches, our NMFN model takes one step further: in the previous approaches the proposed methods are still tied to an underlying graphical model, but in our model we proposed to untie the NMFN model from the underlying graphical model and proposed a few other more aggressive relaxations. The key is our observation about a more explicit connection between iterative inference algorithms and feed-forward neural networks, which makes it clear to see where the restrictions of the model are, and also more straightforward to derive gradients for back-propagation. NMFNs enables some natural relaxations of the restrictions like weight sharing, which leads to faster and better inference as well as more powerful prediction models. When restricting our NMFNs to have the same weights and biases on all layers and tied to the underlying graphical model, we can recover the method in [32, 33] for mean field.

Another work by [65] briefly draws a connection between mean field inference of a specific binary MRF with neural networks, but did not explore further variations.

A few papers have discussed the compatibility between learning and approximate inference algorithms theoretically. [169] shows that inconsistent learning may be beneficial when approximate inference is

used at test time, as long as the learning and test time inference are properly aligned. [89] on the other hand shows that even when using the same approximate inference algorithm at training and test time can have problematic results when the learning algorithm is not compatible with inference. NMFNs do not have this problem, as training follows the exact gradient of the loss function.

On the neural networks side, people have tried to use a neural network to approximate intractable posterior distributions for a long time, especially for learning sigmoid belief networks, see for example [28] and recent paper [122] and citations therein. As far as we know, no previous work on the neural network side have discussed the connection with mean field or belief propagation type methods used for variational inference in graphical models.

A recent paper [83] develops approximate MCMC methods with limited inference budget, which shares the spirit of our work.

5.6 Discussion

The main contribution of this chapter is the theoretical result that connects iterative inference procedures to feed-forward neural networks. On the empirical study side, we have done some preliminary experiments, which indicates that this direction is promising.

From a technical perspective, practically applying the NFMN model in its current form presented in this chapter still faces a number of challenges. First of all, the current NFMN model requires us to prespecify the number of layers to use in the network. The more layers we have in the model, the more capacity there is in the network. On the other hand, having more layers also means the computations are more expensive. The right number of layers to use in the network seems to be model dependent and problem dependent, and may be selected with trial and error.

Second, when we untie the NMFN network parameters from that of the underlying graphical model, all the potentials in the NMFN is also untied and therefore need to be recomputed or updated. This introduces a considerable cost to the inference process. This cost can be alleviated by adapting and untying only parts of the potentials, and share the rest.

Empirically, we found that untying the weights of the NMFN from that of the graphical model can only improve performance by a small amount. Performance upper bound is fundamentally limited by the structure of these networks. In fact, each update in the NMFN as in Eq. 5.3 is very similar to applying a convolution operation to q, while the convolution kernel is very small, usually with a size of 4 (for 4-connected graphs) or 8 (for 8-connected graphs), and on each layer the number of channels must be equal to K the number of output states for each i. Applying modern convolutional neural network design principles, by using larger kernel size, larger number of channels, and strided or dilated convolutions should further improve performance of these models.

Around the same time as our paper on NMFNs [102] or later, a few other researchers have developed ideas similar to ours. [59] extended the idea of unrolling an iterative inference algorithm into a neural network to include belief propagation inference and iterative non-negative matrix factorization. [181] successfully applied similar unrolling idea to fully-connected CRFs and achieved good performance on semantic image segmentation tasks.

This line of work established that models like Neural Mean Field Networks, based on a feed-forward network view of an iterative inference algorithm commonly associated with standard structured output models, have great potential as stand alone structured models. In this chapter, we explored some

of the immediate consequences of realizing the equivalence between inference algorithms and neural networks, and showed that in case of NMFNs, relaxing these restrictions can improve inference efficiency and discriminative performance. This provides a promising new direction of using structured neural networks for solving structured problems, and implies that further relaxation of the restrictions of the network architecture may yield even more performance gains.

The graphical model based approaches and the neural network based approaches handle the structured prediction problems very differently, as briefly discussed in Section 2.3.3. In graphical model based approaches, a model is specified through the definition of the potential functions or scoring functions, and then a separate inference algorithm is applied to make predictions. This separation of modeling and inference is convenient for understanding the model behavior, and makes it easy to change the model and adapt to new needs. However, this separation also causes troubles as inference for complicated models are fundamentally hard, therefore limiting the application of more expressive models. Essentially, the models in graphical model based approaches can be arbitrarily complex, but we are held back by our lack of good inference methods.

The neural network based approaches handle these challenges differently, by directly modeling the computation process of making a prediction. Instead of separating modeling and inference, this approach combines them. Directly modeling the computation process gives us a lot of freedom to design models that have complicated architectures not limited to the ones following a particular inference algorithm derived for a graphical model. From this perspective, these neural network based models can be more powerful than the graphical models as the inference is always efficient (as opposed to intractable inference for graphical models) no matter how complicated the network architecture is.

Chapter 6

Gated Graph Sequence Neural Networks

Many practical applications build on graph-structured data, and thus we often want to perform machine learning tasks that take graphs as inputs. Example applications include predicting properties for molecules in computational chemistry and biology, reasoning with knowledge bases, and predictions for social networks. A few examples are illustrated in Fig. 6.1. Standard approaches to this problem include engineering custom features of an input graph, graph kernels [70, 143], and methods that define graph features in terms of random walks on graphs [131]. More closely related to our goal in this chapter are methods that learn features on graphs, including Graph Neural Networks [45, 140], spectral networks [17] and recent work on learning graph fingerprints for classification tasks on graph representations of chemical molecules [34].

In this chapter, we move one step further along the direction of developing structured neural network models for structured problems, and introduce the Gated Graph Sequence Neural Networks, with its non-sequential variant Gated Graph Neural Networks. These models can exploit the graph structure in structured problems, represent the complicated interactions between entities on the graph, and learn to do various tasks efficiently without getting into the trouble of intractable inference. In this chapter, the input is structured graph data, and the output can be structured as well.

More specifically, our main contribution in this chapter is an extension of Graph Neural Networks [45, 140] that outputs sequences. A secondary contribution is highlighting that Graph Neural Networks and further extensions we develop here are a broadly useful class of neural network model that is applicable to problems currently facing the field. Previous work on feature learning for graph-structured inputs has focused on models that produce single outputs such as graph-level classifications, but many problems with graph inputs require outputting sequences. Examples include paths on a graph, enumerations of graph nodes with desirable properties, or sequences of global classifications mixed with, for example, a start and end node. We are not aware of existing graph feature learning work suitable for this problem. Our motivating application comes from program verification and requires outputting logical formulas, which we formulate as a sequential output problem.

There are two settings for feature learning on graphs: (1) learning a representation of the input graph, and (2) learning representations of the internal state during the process of producing a sequence of outputs. Here, (1) is mostly achieved by previous work on Graph Neural Networks; we make several

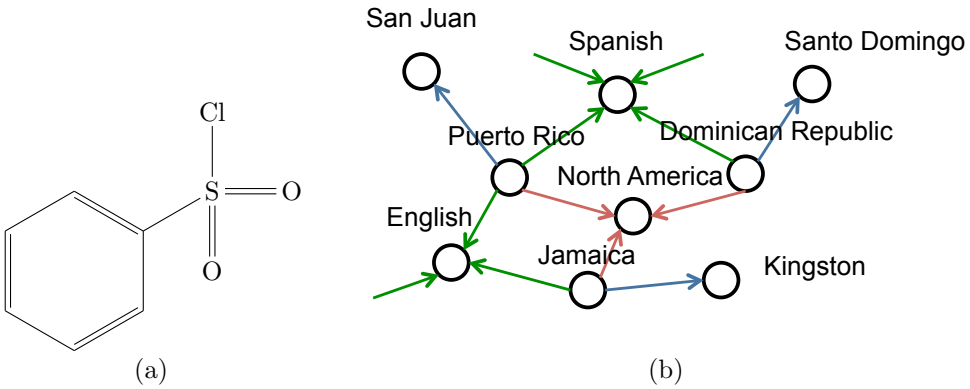


Figure 6.1: Example graph structured data that appear in different applications. (a) predicting properties from molecule structure, here molecules are graphs with atoms being nodes and chemical bounds being edges; (b) reasoning with a knowledge base, where entities are nodes in the graph and relations between entities are edges; in this example, edges with different colors represent different relationships.

adaptations of this framework, including changing it to use modern practices around Recurrent Neural Networks. (2) is important because we desire outputs from graph-structured problems that are not solely individual classifications. In these cases, the challenge is how to learn features on the graph that encode the partial output sequence that has already been produced (e.g., the path so far if outputting a path) and that still needs to be produced (e.g., the remaining path). We will show how the Graph Neural Network framework can be adapted to these settings, leading to a novel graph-based neural network model that we call Gated Graph Sequence Neural Networks (GGS-NNs).

We illustrate aspects of this general model in experiments on bAbI tasks [175] and graph algorithm learning tasks that illustrate the capabilities of the model. Finally, we discuss an application to the verification of computer programs. When attempting to prove properties such as *memory safety* (i.e., that there are no null pointer dereferences in a program), a core problem is to find mathematical descriptions of the data structures used in a program. Following [16], we have phrased this as a machine learning problem where we will learn to map from a set of input graphs representing the state of memory to a logical description of the data structures that have been instantiated. Whereas [16] relied on a large amount of hand-engineering of features, we show that the system can be replaced with a GGS-NN at no cost in accuracy.

6.1 Graph Neural Networks

In this section, we review Graph Neural Networks (GNNs) [45, 140] and introduce notation and concepts that will be used throughout.

GNNs are a general neural network architecture defined according to a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Nodes $v \in \mathcal{V}$ take unique values from $1, \ldots, |\mathcal{V}|$, and edges are pairs $e = (v, v') \in \mathcal{V} \times \mathcal{V}$. We will focus in this chapter on directed graphs, so (v, v') represents a directed edge $v \to v'$, but we note that the framework can easily be adapted to undirected graphs; see [140]. The *node vector* (or *node representation* or *node embedding*) for node v is denoted by $\mathbf{h}_v \in \mathbb{R}^D$. Graphs may also contain node labels $l_v \in \{1, \ldots, L_{\mathcal{V}}\}$ for each node v and edge labels or edge types $l_e \in \{1, \ldots, L_{\mathcal{E}}\}$ for each edge. We will overload notation and let $\mathbf{h}_{\mathcal{S}} = \{\mathbf{h}_v \,|\, v \in \mathcal{S}\}$ when $\mathcal{S}$ is a set of nodes, and $l_{\mathcal{S}} = \{l_e \,|\, e \in \mathcal{S}\}$ when

$\mathcal{S}$ is a set of edges. The function $\textsc{In}(v) = \{v' \,|\, (v', v) \in \mathcal{E}\}$ returns the set of predecessor nodes v' with $v' \to v$. Analogously, $\textsc{Out}(v) = \{v' \,|\, (v, v') \in \mathcal{E}\}$ is the set of successor nodes v' with edges $v \to v'$. The set of all nodes neighboring v is $\textsc{Nbr}(v) = \textsc{In}(v) \cup \textsc{Out}(v)$, and the set of all edges incoming to or outgoing from v is $\textsc{Co}(v) = \{(v', v'') \in \mathcal{E} \,|\, v = v' \vee v = v''\}$.

GNNs map graphs to outputs via two steps. First, there is a propagation step that computes node representations for each node; second, an output model $o_v = g(\mathbf{h}_v, l_v)$ maps from node representations and corresponding labels to an output o_v for each $v \in \mathcal{V}$. In the notation for g, we leave the dependence on parameters implicit, and we will continue to do this throughout. The system is differentiable from end to end, so all parameters are learned jointly using gradient-based optimization.

6.1.1 Propagation Model

Here, an iterative procedure propagates node representations. Initial node representations $\mathbf{h}_v^{(1)}$ are set to arbitrary values, then each node representation is updated following the recurrence below until convergence, where t denotes the timestep:

$$\mathbf{h}_v^{(t)} = f^*(l_v, l_{\text{CO}(v)}, l_{\text{NBR}(v)}, \mathbf{h}_{\text{NBR}(v)}^{(t-1)}).$$

Several variants are discussed in [140] including positional graph forms, node-specific updates, and alternative representations of neighborhoods. Concretely, [140] suggest decomposing $f^*(\cdot)$ to be a sum of per-edge terms:

$$f^*(l_v, l_{\text{CO}(v)}, l_{\text{NBR}(v)}, \mathbf{h}_{\text{NBR}(v)}^{(t)}) = \sum_{v' \in \textsc{In}(v)} f(l_v, l_{(v',v)}, l_{v'}, \mathbf{h}_{v'}^{(t-1)}) + \sum_{v' \in \textsc{Out}(v)} f(l_v, l_{(v,v')}, l_{v'}, \mathbf{h}_{v'}^{(t-1)}),$$

where $f(\cdot)$ is either a linear function of $\mathbf{h}_{v'}$ or a neural network. The parameters of f depends on the configuration of labels, e.g. in the following linear case, $\mathbf{A}$ and $\mathbf{b}$ are learnable parameters and for each tuple of $(l_v, l_{(v',v)}, l_{v'})$ there is a different pair of $\mathbf{A}$ and $\mathbf{b}$ parameters,

$$f(l_v, l_{(v',v)}, l_{v'}, \mathbf{h}_{v'}^{(t)}) = \mathbf{A}^{(l_v, l_{(v',v)}, l_{v'})} \mathbf{h}_{v'}^{(t-1)} + \mathbf{b}^{(l_v, l_{(v',v)}, l_{v'})}.$$

6.1.2 Output Model and Learning

The output model is defined per node and is a differentiable function $g(\mathbf{h}_v, l_v)$ that maps to an output. This is generally a linear or neural network mapping. [140] focus on outputs that are independent per node, which are implemented by mapping the final node representations $\mathbf{h}_v^{(T)}$, to an output $o_v = g(\mathbf{h}_v^{(T)}, l_v)$ for each node $v \in \mathcal{V}$. To handle graph-level classifications, they suggest to create a dummy "super node" that is connected to all other nodes by a special type of edge. Thus, graph-level regression or classification can be handled in the same manner as node-level regression or classification.

Learning is done via the Almeida-Pineda algorithm [1, 132], which has the advantage of not needing to store intermediate states in order to compute gradients. The disadvantage is that parameters must be constrained so that the propagation step is a contraction map. This is needed to ensure convergence, but it may limit the expressivity of the model. When $f(\cdot)$ is a neural network, this is encouraged using a penalty term on the 1-norm of the network's Jacobian.

Contraction Maps Are Bad at Modeling Long Range Dependencies. Here, we show one

example where the intuition why contraction maps have trouble propagating information across a long range in a graph is clear.

Consider a cycle-structured GNN with N nodes $\{1, \ldots, N\}$, where for each node $i > 1$ there is edge $(i-1, i)$ (and there is also an extra edge $(N, 1)$ that completes the cycle). For simplicity we ignored all edge labels and node labels, equivalently this is a simple example with $L_{\mathcal{V}} = 1$ and $L_{\mathcal{E}} = 1$, and we consider the case where each node only has a 1D node representation. At each timestep t we update hidden states $h_1, \ldots, h_N$ as

$$h_i^{(t)} = f(h_{i-1}^{(t-1)}) \tag{6.1}$$

where f is a transformation function that may contain learnable parameters. Let $\varphi(\mathbf{h}) = [\varphi_1(\mathbf{h}), ..., \varphi_N(\mathbf{h})]^\top$, where $\varphi_i(\mathbf{h}^{(t-1)}) = f(h_{i-1}^{(t-1)}) = h_i^{(t)}$.

By definition, φ is contraction map if

$$\|\varphi(\mathbf{h}) - \varphi(\mathbf{h}')\| \leq \rho \|\mathbf{h} - \mathbf{h}'\| \tag{6.2}$$

for a constant $0 \leq \rho < 1$ and any pair $\mathbf{h}, \mathbf{h}'$. Therefore φ being a contraction map implies that each entry of the Jacobian matrix of φ is bounded by ρ, i.e.

$$\left| \frac{\partial \varphi_i}{\partial h_j} \right| \leq \rho, \qquad \forall i, \forall j. \tag{6.3}$$

To see this, consider two vectors $\mathbf{h}$ and $\mathbf{h}'$, where $h_k = h'_k, \forall k \neq j$ and $h_j + \epsilon = h'_j$. The definition in Eq. 6.2 implies that for all i,

$$|\varphi_i(\mathbf{h}) - \varphi_i(\mathbf{h}')| \leq \|\varphi(\mathbf{h}) - \varphi(\mathbf{h}')\| \leq \rho \|\mathbf{h} - \mathbf{h}'\| = \rho|\epsilon|. \tag{6.4}$$

Therefore

$$\left| \frac{\varphi_i(h_1, ..., h_{j-1}, h_j, h_{j+1}, ..., h_N) - \varphi_i(h_1, ..., h_{j-1}, h_j + \epsilon, h_{j+1}, ..., h_N)}{\epsilon} \right| \leq \rho, \tag{6.5}$$

where the left hand side is $\left| \frac{\partial \varphi_i}{\partial h_j} \right|$ by definition as $\epsilon \to 0$.

When $j = i - 1$,

$$\left| \frac{\partial \varphi_i}{\partial h_{i-1}} \right| \leq \rho. \tag{6.6}$$

Also, because of the special cycle graph structure, for all other $j \neq i - 1$ we have $\frac{\partial \varphi_i}{\partial h_j} = 0$. Applying this to the update at timestep t, we get

$$\left| \frac{\partial h_i^{(t)}}{\partial h_{i-1}^{(t-1)}} \right| \leq \rho, \quad \text{and} \quad \left| \frac{\partial h_i^{(t)}}{\partial h_j^{(t-1)}} \right| = 0, \ \forall j \neq i - 1. \tag{6.7}$$

Now let's see how a change in $h_1^{(1)}$ could affect $h_t^{(t)}$. Using the chain rule and the special graph

structure, we have

$$\begin{aligned}\left|\frac{\partial h_t^{(t)}}{\partial h_1^{(1)}}\right| &= \left|\frac{\partial h_t^{(t)}}{\partial h_{t-1}^{(t-1)}} \cdot \frac{\partial h_{t-1}^{(t-1)}}{\partial h_{t-2}^{(t-2)}} \cdots \frac{\partial h_2^{(2)}}{\partial h_1^{(1)}}\right| \\ &= \left|\frac{\partial h_t^{(t)}}{\partial h_{t-1}^{(t-1)}}\right| \cdot \left|\frac{\partial h_{t-1}^{(t-1)}}{\partial h_{t-2}^{(t-2)}}\right| \cdots \left|\frac{\partial h_2^{(2)}}{\partial h_1^{(1)}}\right| \\ &\leq \rho \cdot \rho \cdots \rho = \rho^{t-1}. \end{aligned} \tag{6.8}$$

As $\rho < 1$, this derivative will approach 0 exponentially fast as t grows. Intuitively, this means that the impact one node has on another node far away will decay exponentially, therefore making it difficult to model long range dependencies.

6.2 Gated Graph Neural Networks

We now describe Gated Graph Neural Networks (GG-NNs), our adaptation of GNNs that is suitable for non-sequential outputs. We will describe sequential outputs in the next section. The biggest modification of GNNs is that we use Gated Recurrent Units [26] and unroll the recurrence for a fixed number of steps T and use backpropagation through time in order to compute gradients. This requires more memory than the Almeida-Pineda algorithm, but it removes the need to constrain parameters to ensure convergence. We also extend the underlying representations and output model.

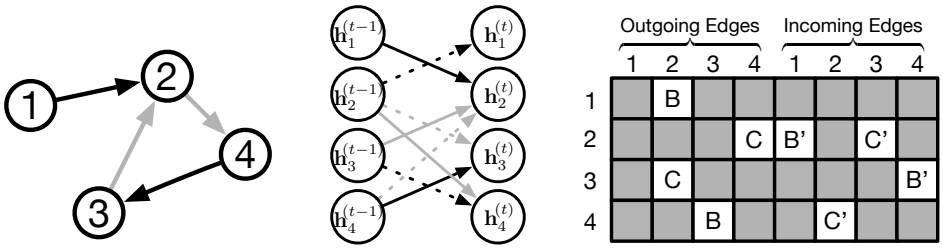
6.2.1 Node Annotations

In GNNs, there is no point in initializing node representations because the contraction map constraint ensures that the fixed point is independent of the initializations. This is no longer the case with GG-NNs, which lets us incorporate node labels as additional inputs. To distinguish these node labels used as inputs from the ones introduced before, we call them *node annotations*, and use vector $\mathbf{x}$ to denote these annotations.

To illustrate how the node annotations are used, consider an example task of training a graph neural network to solve a basic reachability task: whether node t can be reached from node s on a given graph. For this task, there are two problem-related special nodes s and t, and we can set $\mathbf{x}_s = [1, 0]^\top$, $\mathbf{x}_t = [0, 1]^\top$, and $\mathbf{x}_v = [0, 0]^\top$ for any other node v. We then initialize the node state vectors $\mathbf{h}_v^{(1)}$ using these label vectors. The propagation model naturally propagates this task-related information across the graph. In this reachability example, if all nodes reachable from s have their first bit of node representation set to 1 after the propagation, then a classifier can easily tell whether node t is reachable from s using the node representations. In practice, we initialize $\mathbf{h}_v^{(1)}$ by copying $\mathbf{x}_v$ into it and padding with extra 0's to provide more capacity. We do not force $\mathbf{h}_v^{(t)}, t > 1$ to be interpretable, and just let the model learn a proper representation for the tasks from the data.

6.2.2 Propagation Model

The basic recurrence of the propagation model is



(a) (b) (c) $\mathbf{A} = \left[\mathbf{A}^{(\text{out})}, \mathbf{A}^{(\text{in})}\right]$

Figure 6.2: (a) Example graph. Color denotes edge types. (b) Unrolled one timestep. (c) Parameter tying and sparsity in recurrent matrix. Letters denote edge types with B' corresponding to the reverse edge of type B. B and B' denote distinct parameters.

$$\mathbf{h}_v^{(1)} = [\mathbf{x}_v^\top, \mathbf{0}]^\top \tag{6.9}$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top \left[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\right]^\top + \mathbf{b} \tag{6.10}$$

$$\mathbf{z}_v^t = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \tag{6.11}$$

$$\mathbf{r}_v^t = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \tag{6.12}$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right) \tag{6.13}$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}. \tag{6.14}$$

The matrix $\mathbf{A} \in \mathbb{R}^{D|\mathcal{V}| \times 2D|\mathcal{V}|}$ determines how nodes in the graph communicate with each other. The sparsity structure and parameter tying in $\mathbf{A}$ is illustrated in Fig. 6.2. The sparsity structure corresponds to the edges of the graph, and the parameters in each submatrix are determined by the edge type and direction. $\mathbf{A}_{v:} \in \mathbb{R}^{D|\mathcal{V}| \times 2D}$ is the submatrix of $\mathbf{A}$ containing the columns corresponding to node v, both from incoming edges and outgoing edges. Eq. 6.9 is the initialization step, which copies node annotations into the first components of the hidden state and pads the rest with zeros. Eq. 6.10 is the step that passes information between different nodes of the graph via incoming and outgoing edges with parameters dependent on the edge type and direction. $\mathbf{a}_v^{(t)} \in \mathbb{R}^{2D}$ contains activations from edges in both directions. The remaining are GRU-like updates that incorporate information from the other nodes and from the previous timestep to update each node's hidden state. $\mathbf{z}$ and $\mathbf{r}$ are the update and reset gates, $\sigma(x) = 1/(1 + e^{-x})$ is the logistic sigmoid function, and $\odot$ is element-wise multiplication. We initially experimented with a vanilla recurrent neural network-style update, but in preliminary experiments we found this GRU-like propagation step to be more effective.

6.2.3 Output Models

There are several types of one-step outputs that we would like to produce in different situations. First, GG-NNs support *node selection* tasks by making $o_v = g(\mathbf{h}_v^{(T)}, \mathbf{x}_v)$ for each node $v \in \mathcal{V}$ output node scores and applying a softmax over node scores. Second, for graph-level outputs, we define a graph level representation vector as

$$\mathbf{h}_\mathcal{G} = \tanh\left(\sum_{v \in \mathcal{V}} \sigma(i(\mathbf{h}_v^{(T)}, \mathbf{x}_v)) \odot \tanh\left(\mathbf{h}_v^{(T)}\right)\right), \tag{6.15}$$

where $\sigma(i(\mathbf{h}_v^{(T)}, \mathbf{x}_v))$ acts as a soft attention mechanism that decides which nodes are relevant to the current graph-level task. $i(\mathbf{h}_v^{(T)}, \mathbf{x}_v)$ is a neural network that takes the concatenation of $\mathbf{h}_v^{(T)}$ and $\mathbf{x}_v$ as input and outputs real-valued scores. The tanh functions can also be replaced with the identity to get

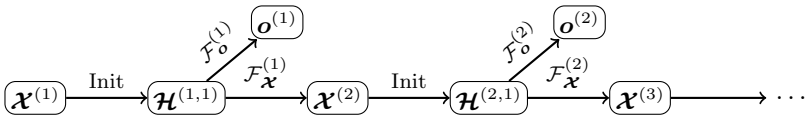


Figure 6.3: Architecture of GGS-NN models.

a simpler version of this graph representation vector.

6.3 Gated Graph Sequence Neural Networks

Here we describe Gated Graph Sequence Neural Networks (GGS-NNs), in which several GG-NNs operate in sequence to produce an output sequence $\boldsymbol{o}^{(1)} \dots \boldsymbol{o}^{(K)}$.

For the k^{th} output step, we denote the matrix of node annotations as $\boldsymbol{\mathcal{X}}^{(k)} = [\mathbf{x}_1^{(k)}; \dots; \mathbf{x}_{|\mathcal{V}|}^{(k)}]^\top \in \mathbb{R}^{|\mathcal{V}| \times L_{\mathcal{V}}}$. We use two GG-NNs $\mathcal{F}_{\boldsymbol{o}}^{(k)}$ and $\mathcal{F}_{\boldsymbol{\mathcal{X}}}^{(k)}$: $\mathcal{F}_{\boldsymbol{o}}^{(k)}$ for predicting $\boldsymbol{o}^{(k)}$ from $\boldsymbol{\mathcal{X}}^{(k)}$, and $\mathcal{F}_{\boldsymbol{\mathcal{X}}}^{(k)}$ for predicting $\boldsymbol{\mathcal{X}}^{(k+1)}$ from $\boldsymbol{\mathcal{X}}^{(k)}$. $\boldsymbol{\mathcal{X}}^{(k+1)}$ can be seen as the states carried over from step k to $k+1$. Both $\mathcal{F}_{\boldsymbol{o}}^{(k)}$ and $\mathcal{F}_{\boldsymbol{\mathcal{X}}}^{(k)}$ contain a propagation model and an output model. In the propagation models, we denote the matrix of node vectors at the t^{th} propagation step of the k^{th} output step as $\boldsymbol{\mathcal{H}}^{(k,t)} = [\mathbf{h}_1^{(k,t)}; \dots; \mathbf{h}_{|\mathcal{V}|}^{(k,t)}]^\top \in \mathbb{R}^{|\mathcal{V}| \times D}$. As before, in step k, we set $\boldsymbol{\mathcal{H}}^{(k,1)}$ by 0-extending $\boldsymbol{\mathcal{X}}^{(k)}$ per node. An overview of the model is shown in Fig. 6.3.

Alternatively, $\mathcal{F}_{\boldsymbol{o}}^{(k)}$ and $\mathcal{F}_{\boldsymbol{\mathcal{X}}}^{(k)}$ can share a single propagation model, and just have separate output models. This simpler variant is faster to train and evaluate, and, in many cases can achieve similar performance level as the full model. But in cases where the desired propagation behavior for $\mathcal{F}_{\boldsymbol{o}}^{(k)}$ and $\mathcal{F}_{\boldsymbol{\mathcal{X}}}^{(k)}$ are different, this variant may not work as well.

We introduce a *node annotation* output model for predicting $\boldsymbol{\mathcal{X}}^{(k+1)}$ from $\boldsymbol{\mathcal{H}}^{(k,T)}$. The prediction is done for each node independently using a neural network $j(\mathbf{h}_v^{(k,T)}, \mathbf{x}_v^{(k)})$ that takes the concatenation of $\mathbf{h}_v^{(k,T)}$ and $\mathbf{x}_v^{(k)}$ as input and outputs a vector of real-valued scores:

$$\mathbf{x}_v^{(k+1)} = \sigma\left(j(\mathbf{h}_v^{(k,T)}, \mathbf{x}_v^{(k)})\right). \tag{6.16}$$

There are two settings for training GGS-NNs: specifying all intermediate annotations $\boldsymbol{\mathcal{X}}^{(k)}$, or training the full model end-to-end given only $\boldsymbol{\mathcal{X}}^{(1)}$, graphs and target sequences. The former can improve performance when we have domain knowledge about specific intermediate information that should be represented in the internal state of nodes, while the latter is more general. We describe both.

Sequence outputs with observed annotations Consider the task of making a sequence of predictions for a graph, where each prediction is only about a part of the graph. In order to ensure we predict an output for each part of the graph exactly once, it suffices to have one bit per node, indicating whether the node has been "explained" so far. In some settings, a small number of annotations are sufficient to capture the state of the output procedure. When this is the case, we may want to directly input this information into the model via labels indicating target intermediate annotations. In some cases, these annotations may be *sufficient*, in that we can define a model where the GG-NNs are rendered conditionally independent given the annotations.

In this case, at training time, given the annotations $\boldsymbol{\mathcal{X}}^{(k)}$ the sequence prediction task decomposes into single step prediction tasks and can be trained as separate GG-NNs. At test time, predicted

annotations from one step will be used as input to the next step. This is analogous to training directed graphical models when data is fully observed.

Sequence outputs with latent annotations More generally, when intermediate node annotations $\boldsymbol{\mathcal{X}}^{(k)}$ are not available during training, we treat them as hidden units in the network, and train the whole model jointly by backpropagating through the whole sequence.

6.4 Explanatory Applications

In this section we present example applications that concretely illustrate the use of GGS-NNs. We focus on a selection of bAbI artificial intelligence (AI) tasks [175] and two graph algorithm learning tasks.

6.4.1 bAbI Tasks

The bAbI tasks are meant to test reasoning capabilities that AI systems should be capable of. In the bAbI suite, there are 20 tasks that test basic forms of reasoning like deduction, induction, counting, and path finding.

We have defined a basic transformation procedure that maps bAbI tasks to prediction problems on graphs suitable for GG-NNs or GGS-NNs. We use the `--symbolic` option from the released bAbI code to get stories that just involve sequences of relations between entities, which are then converted into a graph. Each entity is mapped to a node, and each relation is mapped to an edge with edge label given by the relation type. The full story is consumed and mapped to a single graph. Questions are marked by `eval` in the data and are comprised of a question type (e.g., `has_fear`), and some argument (e.g., one or more nodes). The arguments are converted into initial node annotations, with the i-th bit of the i-th argument node's annotation vector set to 1. For example, if the `eval` line is `eval E > A true`, then `E` gets initial annotation $\mathbf{x}_E^{(1)} = [1, 0]^\top$, `A` gets $\mathbf{x}_A^{(1)} = [0, 1]^\top$, and for all other nodes v, $\mathbf{x}_v^{(1)} = [0, 0]^\top$. Question type is 1 (for '`>`') and output is class 1 (for 'true'). Some tasks have multiple question types, for example Task 4 which has 4 question types: `e`, `s`, `w`, `n`. For such tasks we simply train a separate GG-NN for each task. We do not use the strong supervision labels provided in the dataset, which indicates which line of fact is used for reasoning to get the final answer. We also do not give the GGS-NNs any intermediate annotations (Section 6.3) in any experiments.

While simple, this transformation does not preserve all information about the story (e.g., it discards temporal order of the inputs), and it does not easily handle ternary and higher order relations (e.g., `Yesterday John went to the garden` is not easily mapped to a simple edge, which involves three entities `yesterday`, `John` and `garden`). We also emphasize that it is a non-trivial task to map general natural language to symbolic form[1], so we could not directly apply this approach to arbitrary natural language. Relaxing these restrictions is left for future work.

However, even with this simple transformation, there are a variety of bAbI tasks that can be formulated, including Task 19 (Path Finding), which is arguably the hardest task. We provide baselines to show that the symbolic representation does not help RNNs or LSTMs significantly, and show that GGS-NNs solve the problem with a small number of training instances. We also develop two new bAbI-like tasks that involve outputting sequences on graphs: shortest paths, and a simple form of Eulerian circuits

[1] Although the bAbI data is quite templatic, so it is straightforward to hand-code a parser that will work for the bAbI data; the symbolic option removes the need for this.

(on random connected 2-regular graphs). The point of these experiments is to illustrate the capabilities of GGS-NNs across a variety of problems.

Example 1. As an example, below is an instance from the symbolic dataset for bAbI task 15, Basic Deduction.

```
D is A
B is E
A has_fear F
G is F
E has_fear H
F has_fear A
H has_fear A
C is H
eval B has_fear      H
eval G has_fear      A
eval C has_fear      A
eval D has_fear      F
```

Here the first 8 lines describe the facts, the GG-NN will use these facts to build a graph. Capital letters are nodes, `is` and `has_fear` are interpreted as edge labels or edge types. The last 4 lines are 4 questions asked for this input data. `has_fear` in these lines are interpreted as a question type. For this task, in each question only one node is special, e.g. the `B` in `eval B has_fear`, and we assign a single value 1 to the annotation vector for this special node and 0 to all the other nodes.

For RNN and LSTM the data (for a different example) is converted into token sequences like below:

```
  n6 e1 n1 eol n6 e1 n5 eol n1 e1 n2 eol n4 e1 n5 eol n3 e1 n4 eol n3 e1 n5 eol
n6 e1 n4 eol q1 n6 n2 ans 1
```

where `n<id>` are nodes, `e<id>` are edges, `q<id>` are question types, extra tokens `eol` (end-of-line) and `ans` (answer) are added to give the RNN & LSTM access to the complete information available in the dataset. The final number is the class label.

Example 2. As a second example, below is an instance from the symbolic dataset for bAbI task 19, Path Finding.

```
E s A
B n C
E w F
B w E
eval path B A w,s
```

Here the first 4 lines describe edges, `s`, `n`, `w`, `e` (`e` does not appear in this example) are all different edge types. The last line is a path question, the answer is a sequence of directions `w,s`, as the path going from `B` to `A` is to first go west to `E` then go south to `A`. The `s`, `n`, `w`, `e` in the question lines are treated as output classes.

More Training Details. For all tasks in this section, we generate 1000 training examples and 1000 test examples, 50 of the training examples are used for validation. When evaluating model performance, for all bAbI tasks that contain more than one questions in one example, the predictions for different questions were evaluated independently. As there is randomness in the dataset generation process, we generated 10 such datasets for each task, and report the mean and standard deviation of the evaluation performance across the 10 datasets.

For all explanatory tasks, we start by training different models on only 50 training examples, and gradually increase the number of training examples to 100, 250, 500, and 950 (50 of the training examples are reserved for validation) until the model's test accuracy reaches 95% or above, a success by bAbI standard [175]. For each method, we report the minimum number of training examples it needs to reach 95% accuracy along with the accuracy it reaches with that amount of training examples. In all these cases, we unrolled the propagation process for GG-NNs and GGS-NNs for 5 steps. For bAbI task 4, 15, 16, 18, 19, we used GG-NN with the size of node vectors $\mathbf{h}_v^{(t)}$ set to $D = 4$, $D = 5$, $D = 6$, $D = 3$ and $D = 6$ respectively. For all the GGS-NNs in this section we used the simpler variant in which $\mathcal{F}_{\boldsymbol{o}}^{(k)}$ and $\mathcal{F}_{\boldsymbol{x}}^{(k)}$ share a single propagation model. For shortest path and Eulerian circuit tasks, we used $D = 20$. All models are trained long enough with Adam [71], and the validation set is used to choose the best model to evaluate and avoid models that are overfitting.

Single Step Outputs

We choose four bAbI tasks that are suited to the restrictions described above and require single step outputs: 4 (Two Argument Relations), 15 (Basic Deduction), 16 (Basic Induction), and 18 (Size Reasoning). For Task 4, 15 and 16, a node selection GG-NN is used. For Task 18 we used a graph-level classification version. All the GG-NN networks contain less than 600 parameters[2].

As baselines, we train RNN and LSTM models on the symbolic data in raw sequence form. The RNNs and LSTMs use 50 dimensional embeddings and 50 dimensional hidden layers; they predict a single output at the end of the sequences and the output is treated as a classification problem, the loss is cross entropy. The RNNs and LSTMs contain around 5k and 30k parameters, respectively.

Task	RNN	LSTM	GG-NN
bAbI Task 4	97.3±1.9 (250)	97.4±2.0 (250)	100.0±0.0 (50)
bAbI Task 15	48.6±1.9 (950)	50.3±1.3 (950)	100.0±0.0 (50)
bAbI Task 16	33.0±1.9 (950)	37.5±0.9 (950)	100.0±0.0 (50)
bAbI Task 18	88.9±0.9 (950)	88.9±0.8 (950)	100.0±0.0 (50)

Table 6.1: Accuracy in percentage of different models for different tasks. Number in parentheses is number of training examples required to reach shown accuracy.

[2]For bAbI task 4, we treated 'e', 's', 'w', 'n' as 4 question types and trained one GG-NN for each question type, so strictly speaking for bAbI task 4 our GG-NN model has 4 times the number of parameters of a single GG-NN model. In our experiments we used a GG-NN with 271 parameters for each question type which means 1084 parameters in total.

Test results appear in Table 6.1. For all tasks GG-NN achieves perfect test accuracy using only 50 training examples, while the RNN/LSTM baselines either use more training examples (Task 4) or fail to solve the tasks (Task 15, 16 and 18) even using all the available training examples.

In Table 6.2, we further break down performance of the baselines for task 4 as the amount of training data varies. While both the RNN and LSTM are able to solve the task almost perfectly, the GG-NN reaches 100% accuracy with much less data.

#Training Examples	50	100	250	500	950
RNN	76.7±3.8	90.2±4.0	97.3±1.9	98.4±1.3	99.7±0.4
LSTM	73.5±5.2	86.4±3.8	97.4±2.0	99.2±0.8	99.6±0.8

Table 6.2: Performance breakdown of RNN and LSTM on bAbI task 4 as the amount of training data changes.

Sequential Outputs

The bAbI Task 19 (Path Finding) is arguably the hardest task among all bAbI tasks (see e.g., [152], which reports an accuracy of less than 20% for all methods that do not use the strong supervision). We apply a GGS-NN to this problem, again on the symbolic form of the data (so results are not directly comparable to those in [152]). An extra 'end' class is added to the end of each output sequence; at test time the network will keep making predictions until it predicts the 'end' class.

The results for this task are given in Table 6.3. Both RNN and LSTM fail on this task. However, with only 50 training examples, our GGS-NNs achieve much better test accuracy than RNN and LSTM.

6.4.2 Learning Graph Algorithms

Task	RNN	LSTM	GGS-NNs		
bAbI Task 19	24.7±2.7 (950)	28.2±1.3 (950)	71.1±14.7 (50)	92.5±5.9 (100)	99.0±1.1 (250)
Shortest Path	9.7±1.7 (950)	10.5±1.2 (950)	100.0± 0.0 (50)		
Eulerian Circuit	0.3±0.2 (950)	0.1±0.2 (950)	100.0± 0.0 (50)		

Table 6.3: Accuracy in percentage of different models for different tasks. The number in parentheses is number of training examples required to reach that level of accuracy.

We further developed two new bAbI-like tasks based on algorithmic problems on graphs: Shortest Paths, and Eulerian Circuits. For the first, we generate random graphs and produce a story that lists all edges in the graphs. Questions come from choosing two random nodes A and B and asking for the shortest path (expressed as a sequence of nodes) that connects the two chosen nodes. We constrain the data generation to only produce questions where there is a unique shortest path from A to B of length at least 2. For Eulerian circuits, we generate a random two-regular connected graph and a separate random distractor graph. The question gives two nodes A and B to start the circuit, then the question is to return the Eulerian circuit (again expressed as a sequence of nodes) on the given subgraph that starts by going from A to B. Results are shown in the Table 6.3. RNN and LSTM fail on both tasks, but GGS-NNs learns to make perfect predictions using only 50 training examples.

6.5 Program Verification with GGS-NNs

Our work on GGS-NNs is motivated by a practical application in program verification. A crucial step in automatic program verification is the inference of *program invariants*, which approximate the set of program states reachable in an execution. Finding invariants about data structures is an open problem. As an example, consider the simple `C` function shown below.

```
node* concat(node* a, node* b) {
  if (a == NULL) return b;
  node* cur = a;
  while (cur.next != NULL)
    cur = cur->next;
  cur->next = b;
  return a;
}
```

To prove that this program indeed concatenates the two lists `a` and `b` and that all pointer dereferences are valid, we need to (mathematically) characterize the program's heap in each iteration of the loop. For this, we use *separation logic* [129, 136], which uses *inductive predicates* to describe abstract data structures. For example, a list segment is defined as $\mathsf{ls}(x, y) \equiv x = y \vee \exists v, n.\mathsf{ls}(n, y) * x \mapsto \{\texttt{val} : v, \texttt{next} : n\}$, where $x \mapsto \{\texttt{val} : v, \texttt{next} : n\}$ means that x points to a memory region that contains a structure with `val` and `next` fields whose values are in turn v and n. The $*$ connective is a conjunction as $\wedge$ in Boolean logic, but additionally requires that its operators refer to "separate" parts of the heap. Thus, $\mathsf{ls}(\texttt{cur}, \texttt{NULL})$ implies that `cur` is either `NULL`, or that it points to two values v, n on the heap, where n is described by ls again. The formula $\exists t.\mathsf{ls}(\texttt{a}, \texttt{cur}) * \mathsf{ls}(\texttt{cur}, \texttt{NULL}) * \mathsf{ls}(\texttt{b}, t)$ is an *invariant* of the loop (i.e., it holds when entering the loop, and after every iteration). Using it, we can prove that no program run will fail due to dereferencing an unallocated memory address (this property is called *memory safety*) and that the function indeed concatenates two lists using a Hoare-style verification scheme [62].

The hardest part of this process is coming up with formulas that describe data structures, and this is where we propose to use machine learning. Given a program, we run it a few times and extract the state of memory (represented as a graph; see below) at relevant program locations, and then predict a separation logic formula. Static program analysis tools (e.g., [133]) can check whether a candidate formula is sufficient to prove the desired properties (e.g., memory safety).

6.5.1 Formalization

Representing Heap State as a Graph As inputs we consider directed, possibly cyclic graphs representing the heap of a program. These graphs can be automatically constructed from a program's memory state. Each graph node v corresponds to an address in memory at which a sequence of pointers $v_0, \ldots, v_k$ is stored (we ignore non-pointer values in this work). Graph edges reflect these pointer values, i.e., v has edges labeled with $0, \ldots, k$ that point to nodes $v_0, \ldots, v_k$, respectively. A subset of nodes are labeled as corresponding to program variables.

An example input graph is displayed as "Input" in Fig. 6.4. In it, the node id (i.e., memory address) is displayed in the node. Edge labels correspond to specific fields in the program, e.g., 0 in our example corresponds to the `next` pointer in our example function from the previous section. For binary trees there are two more types of pointers `left` and `right` pointing to the left and right children of a tree node.

Output Representation Our aim is to mathematically describe the shape of the heap. In our model, we restrict ourselves to a syntactically restricted version of separation logic, in which formulas are of the form $\exists x_1, \ldots, x_n . a_1 * \ldots * a_m$, where each atomic formula a_i is either $\mathsf{ls}(x, y)$ (a list from x to y), $\mathsf{tree}(x)$ (a binary tree starting in x), or $\mathsf{none}(x)$ (no data structure at x). Existential quantifiers are used to give names to heap nodes, which are needed to describe a shape, but not labeled by a program variable. For example, to describe a "panhandle list" (a list that ends in a cycle), the first list element on the cycle needs to be named. In separation logic, this can be expressed as $\exists t . \mathsf{ls}(x, t) * \mathsf{ls}(t, t)$.

Data We can generate synthetic (labeled) datasets for this problem. For this, we fix a set of predicates such as ls and tree (extensions could consider doubly-linked list segments, multi-trees, ...) together with their inductive definitions. Then we enumerate separation logic formulas instantiating our predicates using a given set of program variables. Finally, for each formula, we enumerate heap graphs satisfying that formula. The result is a dataset consisting of pairs of heap graphs and associated formulas that are used by our learning procedures.

6.5.2 Formulation as GGS-NNs

It is easy to obtain the node annotations for the intermediate prediction steps from the data generation process. So we train a variant of GGS-NN with observed annotations (observed at training time; not test time) to infer formulas from heap graphs. Note that it is also possible to use an unobserved GGS-NN variant and do end-to-end learning. The procedure breaks down the production of a separation logic formula into a sequence of steps. We first decide whether to declare existential variables, and if so, choose which node corresponds to the variable. Once we have declared existentials, we iterate over all variable names and produce a separation logic formula describing the data structure rooted at the node corresponding to the current variable.

The full algorithm for predicting separation logic formula appears below, as Alg. 1. We use three explicit node annotations, namely *is-named* (heap node labeled by program variable or declared existentially quantified variable), *active* (cf. algorithm) and *is-explained* (heap node is part of data structure already predicted). Initial node labels can be directly computed from the input graph: "is-named" is on for nodes labeled by program variables, "active" and "is-explained" are always off (done in line 2). The commented lines in the algorithm are implemented using a GG-NN, i.e., Alg. 1 is an instance of our GGS-NN model. An illustration of the beginning of a run of the algorithm is shown in Fig. 6.4, where each step is related to one line of the algorithm.

6.5.3 Model Setup Details

We use the full GGS-NN model where $\mathcal{F}_{\boldsymbol{o}}^{(k)}$ and $\mathcal{F}_{\boldsymbol{\mathcal{X}}}^{(k)}$ have separate propagation models. For all the GG-NN components in the GGS-NN pipeline, we unrolled the propagation process for 10 time steps. The GGS-NNs associated with step (†) (deciding whether more existentially quantified variable need to be declared) and (‡) (identify which node need to be declared as existentially quantified) uses $D = 16$

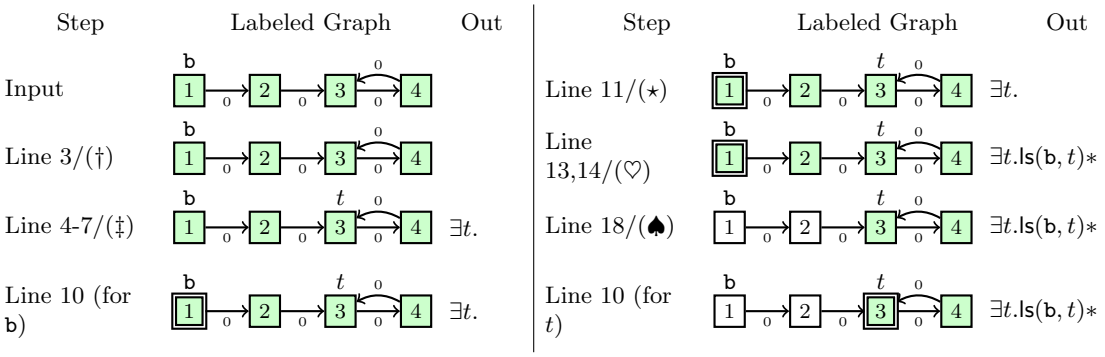


Figure 6.4: Illustration of the first 8 steps to predict a separation logic formula from a memory state. Label *is-named* signified by variable near node, *active* by double border, *is-explained* by white fill.

Algorithm 1 Separation logic formula prediction procedure

Input: Heap graph $\mathcal{G}$ with named program variables

$\boldsymbol{\mathcal{X}} \leftarrow$ compute initial labels from $\mathcal{G}$
$\boldsymbol{\mathcal{H}} \leftarrow$ initialize node vectors by 0-extending $\boldsymbol{\mathcal{X}}$
while $\exists$ quantifier needed **do** ▷ **Graph-level Classification (†)**
  $t \leftarrow$ fresh variable name
  $v \leftarrow$ pick node ▷ **Node Selection (‡)**
  $\boldsymbol{\mathcal{X}} \leftarrow$ turn on "is-named" for v in $\boldsymbol{\mathcal{X}}$
  print "$\exists t.$"
end while
for node v_ℓ with label "is-named" in $\boldsymbol{\mathcal{X}}$ **do**
  $\boldsymbol{\mathcal{H}} \leftarrow$ initialize node vectors, turn on "active" label for v_ℓ in $\boldsymbol{\mathcal{X}}$
  $pred \leftarrow$ pick data structure predicate ▷ **Graph-level Classification (⋆)**
  if $pred = \mathsf{ls}$ **then**
    $\ell_{end} \leftarrow$ pick list end node ▷ **Node Selection (♡)**
    print "$\mathsf{ls}(\ell, \ell_{end})\ *$"
  else
    print "$pred(\ell)\ *$"
  end if
  $\boldsymbol{\mathcal{X}} \leftarrow$ update node annotations in $\boldsymbol{\mathcal{X}}$ ▷ **Node Annotation (♠)**
end for

dimensional node representations. For all other GGS-NN components, $D = 8$ is used. Adam [71] is used for optimization; the models are trained on minibatches of 20 graphs, and optimized until training error is very low. For the graph-level classification tasks, we also artificially balanced classes to have even number of examples from each class in each minibatch. All the GGS-NN components contain less than 5k parameters and no overfitting is observed during training.

6.5.4 Batch Prediction Details

In practice, a set of heap graphs will be given as input and a single output formula is expected to describe and be consistent with all the input graphs. The different heap graphs can be snapshots of the heap state at different points in the program execution process, or different runs of the same program with different inputs. We call this the "batch prediction" setup contrasting with the single graph prediction described in the main paper.

To make batch predictions, we run one GGS-NN for each graph simultaneously. For each prediction step, the outputs of all the GGS-NNs at that step across the batch of graphs are aggregated.

For node selection outputs, the common named variables link nodes on different graphs together, which is the key for aggregating predictions in a batch. We compute the score for a particular named variable t as $o_t = \sum_g o^g_{\mathcal{V}_g(t)}$, where $\mathcal{V}_g(t)$ maps variable name t to a node in graph g, and $o^g_{\mathcal{V}_g(t)}$ is the output score for named variable t in graph g. When applying a softmax over all names using o_t as scores, this is equivalent to a model that computes $p(toselect = t) \propto \prod_g p_g(toselect = \mathcal{V}_g(t))$.

For graph-level classification outputs, we add up scores of a particular class across the batch of graphs, or equivalently compute $p(class = k) \propto \prod_g p_g(class = k)$. Node annotation outputs are updated for each graph independently as different graphs have completely different set of nodes. However, when the algorithm tries to update the annotation for one named variable, the nodes associated with that variable in all graphs are updated. During training, all labels for intermediate steps are available to us from the data generation process, so the training process again can be decomposed to single output single graph training.

A more complex scenario allowing for nested data structures (e.g., list of lists) was discussed in [16]. We have also successfully extended the GGS-NN model to this case.

6.5.5 Experiments

For this paper, we produced a dataset of 327 formulas that involves three program variables, with 498 graphs per formula, yielding around 160,000 formula/heap graph combinations. To evaluate, we split the data into training, validation and test sets using a 6:2:2 split on the formulas (i.e., the formulas in the test set were not in the training set). We measure correctness by whether the formula predicted at test time is logically equivalent to the ground truth; equivalence is approximated by canonicalizing names and order of the formulas and then comparing for exact equality.

We compared our GGS-NN-based model with a method we developed earlier [16]. The earlier approach treats each prediction step as standard classification, and requires complex, manual, problem-specific feature engineering, to achieve an accuracy of 89.11%. In contrast, our new model was trained with no feature engineering and very little domain knowledge and achieved an accuracy of 89.96%.

An example heap graph and the corresponding separation logic formula found by our GGS-NN model is shown in Fig. 6.5. This example also involves nested data structures and the batching extension

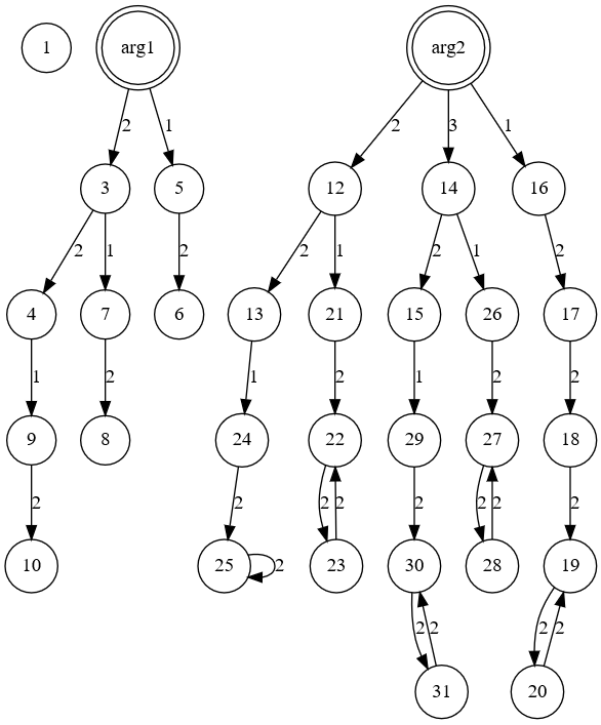


Figure 6.5: A heap graph example that contains two named variables `arg1` and `arg2`, and one isolated `NULL` node (node 1). All the edges to `NULL` are not shown here for clarity. The numbers on edges indicate different edge types. Our GGS-NN model successfully finds the right formula $\mathsf{ls}(\texttt{arg1}, \texttt{NULL}, \lambda t_1 \rightarrow \mathsf{ls}(t_1, \texttt{NULL}, \top)) * \mathsf{tree}(\texttt{arg2}, \lambda t_2 \rightarrow \exists e_1.\mathsf{ls}(t_2, e_1, \top) * \mathsf{ls}(e_1, e_1, \top))$.

developed in the previous section.

We have also successfully used our new model in a program verification framework, supplying needed program invariants to a theorem prover to prove correctness of a collection of list-manipulating algorithms such as insertion sort. The following Table 6.4 lists a set of benchmark list manipulation programs and the separation logic formula invariants found by the GGS-NN model, which were successfully used in a verification framework to prove the correctness of corresponding programs. A further extension of the current pipeline has been shown to be able to successfully prove more sophisticated programs like sorting programs and various other list-manipulating programs.

6.6 Related Work

The most closely related work is GNNs, which we have discussed at length above. [118] proposed another closely related model that differs from GNNs mainly in the output model. GNNs have been applied in several domains [45, 30, 140, 163], but they do not appear to be in widespread use in our community. Part of our aim here is to publicize GNNs as a useful and interesting neural network variant.

An analogy can be drawn between our adaptation from GNNs to GG-NNs, to the work of [32] and [150] in the structured prediction setting. There belief propagation (which must be run to near convergence to get good gradients) is replaced with truncated belief propagation updates, and then the model is trained so that the truncated iteration produce good results after a fixed number of iterations.

Program	Invariant Found
Traverse1	$\mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$
Traverse2	$\mathtt{curr} \neq \mathtt{NULL} * \mathtt{lst} \neq \mathtt{NULL} * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$
Concat	$\mathtt{a} \neq \mathtt{NULL} * a \neq b * b \neq \mathtt{curr} * \mathtt{curr} \neq \mathtt{NULL}$ $* \mathsf{ls}(\mathtt{curr}, \mathtt{NULL}) * \mathsf{ls}(a, \mathtt{curr}) * \mathsf{ls}(b, \mathtt{NULL})$
Copy	$\mathsf{ls}(\mathtt{curr}, \mathtt{NULL}) * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{cp}, \mathtt{NULL})$
Dispose	$\mathsf{ls}(\mathtt{lst}, \mathtt{NULL})$
Insert	$\mathtt{curr} \neq \mathtt{NULL} * \mathtt{curr} \neq \mathtt{elt} * \mathtt{elt} \neq \mathtt{NULL} * \mathtt{elt} \neq \mathtt{lst} * \mathtt{lst} \neq \mathtt{NULL}$ $* \mathsf{ls}(\mathtt{elt}, \mathtt{NULL}) * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$
Remove	$\mathtt{curr} \neq \mathtt{NULL} * \mathtt{lst} \neq \mathtt{NULL} * \mathsf{ls}(\mathtt{lst}, \mathtt{curr}) * \mathsf{ls}(\mathtt{curr}, \mathtt{NULL})$

Table 6.4: Example list manipulation programs and the separation logic formula invariants the GGS-NN model founds from a set of input graphs. The "$\neq$" parts are produced by a deterministic procedure that goes through all the named program variables in all graphs and checks for inequality.

Similarly, Recursive Neural Networks [43, 146] being extended to Tree LSTMs [155] is analogous to our using of GRU updates in GG-NNs instead of the standard GNN recurrence with the aim of improving the long-term propagation of information across a graph structure.

The general idea expressed in this chapter of assembling problem-specific neural networks as a composition of learned components has a long history, dating back at least to the work of [60] on assembling neural networks according to a family tree structure in order to predict relations between people. Similar ideas appear in [52] and [12].

Graph kernels [143, 70] can be used for a variety of kernel-based learning tasks with graph-structured inputs, but we are not aware of work that learns the kernels and outputs sequences. [131] convert graphs into sequences by following random walks on the graph then learns node embeddings using sequence-based methods. [149] map graphs to graph vectors then classify using an output neural network. There are several models that make use of similar propagation of node representations on a graph structure. [17] generalize convolutions to graph structures. The difference between their work and GNNs is analogous to the difference between convolutional and recurrent networks. [34] also consider convolutional like operations on graphs, building a learnable, differentiable variant of a successful graph feature. Other graph convolutional neural networks work like [73, 3] can be thought of as simplications of our model, which may be suitable when overfitting is an important issue in the problem domain. [112] converts an arbitrary undirected graph to a number of different DAGs with different orientations and then propagates node representations inwards towards each root, training an ensemble of models. In all of the above, the focus is on one-step problems.

GNNs and our extensions have many of the same desirable properties of pointer networks [168]; when using node selection output layers, nodes from the input can be chosen as outputs. There are two main differences: first, in GNNs the graph structure is explicit, which makes the models less general but may provide stronger generalization ability; second, pointer networks require that each node has properties (e.g., a location in space), while GNNs can represent nodes that are defined only by their position in the graph, which makes them more general along a different dimension.

GGS-NNs are related to soft alignment and attentional models (e.g., [4, 90, 152]) in two respects: first, the graph representation in Eq. 6.15 uses context to focus attention on which nodes are important to the current decision; second, node annotations in the program verification example keep track of which nodes have been explained so far, which gives an explicit mechanism for making sure that each node in

the input has been used over the sequence of producing an output.

6.7 Discussion

What is being learned? It is instructive to consider what is being learned by the GG-NNs. To do so, we can draw analogy between how the bAbI task 15 would be solved via a logical formulation. As an example, consider the subset of lines needed to answer one example question shown below.

```
B is E
E has_fear H
eval B has_fear
```

To do logical reasoning, we would need not only a logical encoding of the facts present in the story but also the background world knowledge encoded as inference rules such as

$$\texttt{is(x, y)} \wedge \texttt{has-fear(y, z)} \implies \texttt{has-fear(x, z)}. \tag{6.17}$$

A good model has to understand that `is` indicates equivalence and `has-fear` allows the relation to be transferred to equivalent entities, which is non-trivial knowledge that needs to be learned from data. Our encoding of the tasks simplifies the parsing of the story into graph form, but it does not provide any of the background knowledge. The GG-NN model can be seen as learning this, with results stored in the neural network weights.

Discussion The results in this chapter show that GGS-NNs have desirable inductive biases across a range of problems that have some intrinsic graph structure to them, and we believe there to be many more cases where GGS-NNs will be useful. There are, however, some limitations that need to be overcome to make them apply even more broadly. Two limitations that we mentioned previously are that the bAbI task translation does not incorporate temporal order of inputs or ternary and higher order relations. We can imagine several possibilities for lifting these restrictions, such as concatenating a series of GG-NNs, where there is one GG-NNs for each edge, and representing higher order relations as factor graphs. A more significant challenge is how to handle less structured input representations. For example, in the bAbI tasks it would be desirable not to use the symbolic form of the inputs.

The current GGS-NNs formulation specifies a question only after all the facts have been consumed. This implies that the network must try to derive all consequences of the seen facts and store all pertinent information to a node within its node representation. This is likely not ideal; it would be preferable to develop methods that take the question as an initial input, and then dynamically derive the facts needed to answer the question.

The general graph structure learning task for GG-NNs and GGS-NNs can potentially be done with a model that processes raw input data. There are two key steps for building a graph: creating graph nodes and adding edges. Creating graph nodes is about recognizing entities. For language understanding this step may be done by a range of entity recognition methods. Adding edges is about recognizing the relationship between entities. A straightforward method to do this is to use an RNN model to read in

natural language sentences, and predict if a sentence indicates any relationships between entities. Better methods to solve these problems may involve some attention mechanism.

From our experience experimenting with the GG-NNs and GGS-NNs, we noted that since we always unroll the graph propagation process for a fixed number of T steps, it is likely that the models will learn to overfit to this particular number of steps. If the learned models are tested on graphs of vastly different sizes, then the model will fail in unexpected ways. To avoid this problem, we can vary the number of propagation steps during training, by either stochastically sample a T, or choose T based on graph size. Choosing just the right T is nontrivial, but choosing a slightly larger than ideal T is relatively easy. If some noise is added to T, the model will learn to stop propagation after a certain number of steps, which is helpful if at test time T is not chosen properly. By using these techniques, it is possible to learn models on small graphs and apply to graphs of very different sizes.

Another challenge in applying these models to large scale problems is scalability. Big knowledge bases may have millions or billions of entities, and even more relations. With graphs at this scale, it is impractical and unnecessary to propagate information around the full graph. Local computations may be good enough for this purpose, where the information is propagated only for a limited number of steps, and then when making a prediction, information is also aggregated only in the local neighborhood. Parallelization is necessary for speeding up the more global computations that have to aggregate information from the whole graph. The propagation process can be easily parallelized, as in each propagation step each node propagates its state to its neighbors independently.

We are optimistic about further applications of GGS-NNs. We are particularly interested in continuing to develop end-to-end learnable systems that can learn about semantic properties of programs, that can learn more complicated graph algorithms, and in applying these ideas to problems that require reasoning over knowledge bases and databases. More generally, we consider these graph neural networks as representing a step towards a model that can combine structured representations with the powerful algorithms of deep learning, with the aim of taking advantage of known structure while learning and inferring how to reason with and extend these representations.

Thinking along this direction, it is interesting to compare in particular GG-NNs with the Neural Mean Field Networks and graphical model based medthods for structured problems presented in earlier chapters. In fact, the GG-NNs model can be directly applied to many structured prediction problems normally solved with graphical models. For example, in image segmentation problems, it is possible to build a neighborhood graph and have each pixel be a node in the graph, and then run a GG-NNs model to make a per node prediction. Compared to the graphical model based approaches, the prediction process of GG-NNs is not restricted to follow a particular inference algorithm derived for a graphical model, and arbitrary nonlinearities and network architectures can all be used for making predictions. As an example, we used the gating mechanism in our models and found it to increase the model capacity significantly.

After learning a GG-NNs or GGS-NNs model, another particularly interesting thing to do is to recover the knowledge learned by the network. As mentioned earlier, when used for reasoning, our model can learn the inference rules like the one shown in Eq. 6.17. However, this knowledge is hidden in the network weights. Extracting such explicit inference rules is helpful for understanding why the network makes a certain prediction, and is also helpful for debugging. These inference rules themselves are also useful, and may be used to form part of a knowledge base. There are a number of challenges for extracting such explicit inference rules. First, the state vectors are usually not interpretable, making

it hard to associate the state vectors with explicit reference to properties. Second the network weights are usually not interpretable either. More interpretability may be added by regularizing the network weights and state vectors to be sparse.

Chapter 7

Conclusion

In this thesis, I presented a collection of work done during my Ph.D. studies with the goal of building more expressive structured models. Structures are very common in data and in the problems that machine learning and related communities are more and more interested in. Exploiting the structures and building expressive structured models is the key to efficiently utilizing the data and handling the increasing complexity in the models and tasks.

7.1 Summary

This thesis started with a discussion of structured models by presenting the standard structured output models and emphasized the key challenge unique to structured output problems, namely the hard inference problem. The hardness of inference leads to the key consideration of balancing model complexity and inference complexity in building structured output problems.

Exploring tractable high-order models is one way to extend the expressive power of such standard structured output models. Chapter 3 presented our contribution on compositional high-order patten potentials, which is a step in this direction that increases the expressive power of structured output models, while at the same time efficient inference algorithms are developed by exploiting the structure of these high-order potentials.

Another challenge unique to structured output problems is the difficulty of obtaining accurate labels, which is a result of the complicated output space. Training good structured models thus requires much more labeling effort than normal unstructured tasks. Our work on semi-supervised learning of structured models presented in Chapter 4 provides one way to solve this problem. The key to the success of this method is to find a good model for the similarity between examples and build a rich enough high order model to handle the propagation of label information based on this similarity metric.

However, the standard structured output models are still fundamentally limited by the hardness of the inference problem in complicated models, making the effort to extend such models to complex tasks more and more challenging. Our work on Neural Mean Field Networks presented in Chapter 5 and the related line of work reveal that these structured output models are equivalent to a constrained type of neural network model, thus providing a promising new direction to explore, which builds expressive structured neural network models directly.

The benefit of structured neural network models is that they are built to be inference models that

directly make predictions by computation. This contrasts them with standard structured output models that rely on a separate inference process which requires solving a usually intractable optimization problem. This separation of modeling and inference in standard structured output models and graphical models is nice to have as it leads to a more modular design of the models, but potentially causes trouble for inference. Neural networks take a more integrated approach and give us more freedom to design complicated network architectures to increase the capabilities of the models.

The gated graph sequence neural networks and gated graph neural networks presented in Chapter 6 is an initial attempt to build such structured neural networks, and the results are promising. These models are particularly good for graph structured data, and perform reasoning on the graph by propagating information along the graph edges. In the applications these models achieved great performance for program verification and natural language reasoning tasks.

7.2 Future Directions

There are reasons to believe that machine learning research will be moving towards structured models more and more in the future, as the whole community moves towards solving more and more challenging problems. More complexity of the problems requires increasingly more complicated models, and exploiting structure of the problems and data is a nice way to handle complexity, if not the only way. On the other side, well studied standard problems like classification have almost reached a level of saturation, where machine learning methods achieved super human performance [55, 64, 56] on even the largest academic datasets like ImageNet [29] which has millions of images and thousands of classes.

For standard structured output models and graphical model based methods, the key challenge in building more expressive models is always the balance between model complexity and inference complexity. Future research in this direction should focus on either studying better inference techniques for existing models, or developing new models for which inference can be done efficiently. Exploring better inference algorithms may improve upon existing approaches, but making significant progress is extremely difficult due to the intractability of inference even for very simple models. Developing new models with efficient inference is more promising because we have larger freedom to explore.

Latent variable models are especially appealing for this purpose. Such models decompose high order interactions into interactions with latent variables, therefore utilizing more structure in the model, and have potential to make inference easier. Our CHOPPs model presented in Chapter 3 is one example of these latent variable models. The space of possible such models is big enough to be explored a lot more. Coming up with new latent variable model architectures requires careful design such that these models can capture the interactions between variables well. Learning in latent variable models is usually done through an EM-like alternating optimization procedure; when the posterior over latent variables is intractable, a variational bound can be used instead for optimization. Improving these learning procedures is also an interesting direction to explore. Recently neural network based posterior inference methods have emerged as a successful approach for handling complex posterior distributions, and they may be successfully applied here as well.

On the application side, the standard structured output models may be greatly improved by using better models for the potential functions. Recent trend of using deep neural networks to model the potential functions, in particular the unary and pairwise potentials, in structured output problems has led to rapid improvement of performance for many different tasks. Extending neural network models to

high order potentials may further improve performance.

On the other hand, there are still common structured problems that cannot be readily handled by existing structured output models. For example, in object detection problems, one important step in the pipeline is nonmaximal suppression, i.e. keeping the detection with the highest score, and suppressing all others in a local neighborhood. This nonmaximal suppression step is essentially a structured prediction problem, where the input can be sets of detection proposals, and the output is a set of consistent detections with false positives and nonmaximal detections filtered out. This step is critical for object detection, but usually it is still done in a heuristic way. This is a challenging problem as (1) the space of possible detection proposals is huge, and representing a collection of detections is nontrivial; (2) defining a criterion for choosing a set of detections from a pool of candidates is nontrivial. Similar examples can be found in other application domains. Particularly, anywhere post-processing is still actively being used has potential to be improved upon with a model that properly handles the structure of the problem. Other than these, applying existing structured models, like sequence models, 2D models and graph models, to new application domains usually encounters new challenges and requires some non-trivial adaptation or novel architecture changes. Developing structured output models for these problems requires smart design, but also are unique opportunities to greatly expand the applicability of structured models.

Neural network models do not have the intractable inference problem, and therefore we have a lot of freedom to design network architectures. The key challenge to apply such models to a wider range of tasks is the design of problem dependent network architecture. Bootstrapping from an established graphical model and then designing neural network architecture to be close to an unrolled inference process is a viable option, but more significant gains can be potentially obtained with more customized architectures that go beyond the existing graphical model architectures.

Similar problem of designing the right architecture to handle structure appears when using neural networks to model structured input data as well. RNNs are good at modeling sequence data, CNNs are good at modeling spatial data, and the graph neural networks and our developments in Chapter 6 are good at handling graph-structured data. But there are still a range of other forms of input currently cannot be handled by neural networks directly, like matchings and rankings.

Current neural network based models do not share a number of nice properties of standard structured output models based on graphical models: (1) it is relatively easier to incorporate prior knowledge into standard structured output models than neural network models, by simply adding more terms in the scoring function which specifies interpretable constraints on the solutions; changing the loss function to incorporate desired prior is an alternative way to pass on knowledge from the prior to the model, which can be utilized by both the neural network models and the standard structured output models, but it is not as straightforward and predictable as adding terms to the scoring function; (2) the standard structured output models are much more explainable than neural network models; (3) the behavior of these standard structured output models are more predictable and many have well studied guarantees. The research in developing structured neural network models, once reaching a satisfactory level of performance, may move toward achieving these properties, as they are useful for practitioners to adopt these models and make neural network models more transparent.

Neural network models in general have their own unique challenges, notably the learning process is usually a very challenging nonlinear optimization problem. Also to train a good neural network model normally requires more data than simple models. Efficiently exploiting the structure in the problems

may help alleviate both problems.

Among all these future directions, I think developing new structured neural network models, like RNNs, CNNs, graph neural nets and others, and expanding the applicability of structured models to cover more applications are potentially the two most fruitful future directions to explore. As our research community moves toward more complex and challenging tasks, I expect to see more and more structured models to be studied, adopted and used in practice.

Bibliography

[1] Luis B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks*, pages 102–111. IEEE Press, 1990.

[2] Yasemin Altun, David McAllester, and Mikhail Belkin. Maximum margin semi-supervised learning for structured variables. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[3] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. 2015.

[6] David Belanger and Andrew McCallum. Structured prediction energy networks. In *International Conference on Machine Learning (ICML)*, 2016.

[7] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research (JMLR)*, 2006.

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research (JMLR)*, 3(Feb):1137–1155, 2003.

[9] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[10] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Conference on Learning Theory (COLT)*, 1998.

[11] Eran Borenstein and Shimon Ullman. Class-specific, top-down segmentation. In *European Conference on Computer Vision (ECCV)*, 2002.

[12] Léon Bottou. From machine learning to machine reasoning. *Machine learning*, 94(2):133–149, 2014.

[13] 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.

[14] Yuri Y. Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *IEEE International Conference on Computer Vision (ICCV)*, 2001.

[15] Ulf Brefeld and Tobias Scheffer. Semi-supervised learning for structured output variables. In *International Conference on Machine Learning (ICML)*, 2006.

[16] Marc Brockschmidt, Yuxin Chen, Byron Cook, Pushmeet Kohli, and Daniel Tarlow. Learning to decipher the heap for program verification. In *Workshop on Constructive Machine Learning at the International Conference on Machine Learning (CMLICML)*, 2015.

[17] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[18] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning (ICML)*, 2005.

[19] Andrew Carlson, Justin Betteridge, Richard C Wang, Estevam R Hruschka Jr, and Tom M Mitchell. Coupled semi-supervised learning for information extraction. In *ACM International Conference on Web Search and Data Mining (WSDM)*, 2010.

[20] Joao Carreira and Cristian Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3241–3248. IEEE, 2010.

[21] Joao Carreira, Rui Caseiro, Jorge Batista, and Cristian Sminchisescu. Semantic segmentation with second-order pooling. In *European Conference on Computer Vision (ECCV)*, pages 430–443. Springer, 2012.

[22] Ming-Wei Chang, Lev Ratinov, and Dan Roth. Guiding semi-supervision with constraint-driven learning. In *Annual Meeting of the Association of Computational Linguistics (ACL)*, 2007.

[23] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. *Semi-supervised learning*. MIT press Cambridge, 2006.

[24] 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 *International Conference on Learning Representations (ICLR)*, 2015.

[25] Liang-Chieh Chen, Alexander G Schwing, Alan L Yuille, and Raquel Urtasun. Learning deep structured models. In *International Conference on Machine Learning (ICML)*, 2015.

[26] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[27] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[28] Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

[29] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[30] Vincenzo Di Massa, Gabriele Monfardini, Lorenzo Sarti, Franco Scarselli, Marco Maggini, and Marco Gori. A comparison between recursive neural networks and graph neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, 2006.

[31] Xuetao Ding, Xiaoming Jin, Yujia Li, and Lianghao Li. Celebrity recommendation with collaborative social topic regression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[32] Justin Domke. Parameter learning with truncated message-passing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2937–2943, 2011.

[33] Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2013.

[34] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[35] S.M. Ali Eslami and Christopher K.I. Williams. A generative model for parts-based object segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 100–107, 2012.

[36] S.M. Ali Eslami, Nicolas Heess, Christopher K.I. Williams, and John Winn. The shape boltzmann machine: a strong model of object shape. *International Journal of Computer Vision (IJCV)*, 107 (2):155–176, 2014.

[37] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338, 2010.

[38] Yoav Freund and David Haussler. Unsupervised learning of distributions of binary vectors using 2-layer networks. In *Advances in Neural Information Processing Systems (NIPS)*, 1991.

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

[40] Kuzman Ganchev, Joao Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research (JMLR)*, 2010.

[41] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, (6):721–741, 1984.

[42] Amir Globerson and Tommi S. Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *Advances in Neural Information Processing Systems*, pages 553–560, 2008.

[43] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks*, volume 1, pages 347–352. IEEE, 1996.

[44] Josep M Gonfaus, Xavier Boix, Joost Van de Weijer, Andrew D Bagdanov, Joan Serrat, and Jordi Gonzalez. Harmony potentials for joint classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3280–3287. IEEE, 2010.

[45] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *International Joint Conference on Neural Networks (IJCNN)*, 2005.

[46] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[47] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2006.

[48] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE international conference on acoustics, speech and signal processing*, 2013.

[49] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

[50] Varun Gulshan, Carsten Rother, Antonio Criminisi, Andrew Blake, and Andrew Zisserman. Geodesic star convexity for interactive image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[51] Rahul Gupta, Ajit A Diwan, and Sunita Sarawagi. Efficient inference with cardinality-based clique potentials. In *International Conference on Machine Learning (ICML)*, pages 329–336, 2007.

[52] Barbara Hammer and Brijnesh J. Jain. Neural methods for non-standard data. In *European Symposium on Artificial Neural Networks (ESANN)*, 2004.

[53] W. Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[54] Tamir Hazan and Raquel Urtasun. A primal-dual message-passing algorithm for approximated large scale structured prediction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 838–846, 2010.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[57] Luheng He, Jennifer Gillenwater, and Ben Taskar. Graph-based posterior regularization for semi-supervised structured prediction. In *Conference on Computational Natural Language Learning (CoNLL)*, 2013.

[58] Xuming He, Richard S. Zemel, and Miguel Á Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[59] John R Hershey, Jonathan Le Roux, and Felix Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.

[60] Geoffrey E. Hinton. Representing part-whole hierarchies in connectionist networks. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pages 48–54. Erlbaum., 1988.

[61] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 2002.

[62] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[63] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[64] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[65] Viren Jain, Joseph F. Murray, Fabian Roth, Srinivas Turaga, Valentin Zhigulin, Kevin L. Briggman, Moritz N. Helmstaedter, Winfried Denk, and H. Sebastian Seung. Supervised learning of image restoration with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2007.

[66] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993.

[67] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning (ICML)*, 1999.

[68] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning (ICML)*, 2015.

[69] Andrew Kae, Kihyuk Sohn, Honglak Lee, and Erik Learned-Miller. Augmenting crfs with boltzmann machine shape priors for image labeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[70] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning (ICML)*, 2003.

[71] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[72] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2013.

[73] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[74] Ryan Kiros, Ruslan Salakhutdinov, and Richard S. Zemel. Multimodal neural language models. In *International Conference on Machine Learning (ICML)*, 2014.

[75] Jyri J. Kivinen and Christopher K.I. Williams. Multiple texture boltzmann machines. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[76] Pushmeet Kohli, Lu'bor Ladický, and Philip H.S. Torr. Robust higher order potentials for enforcing label consistency. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[77] Pushmeet Kohli, Anton Osokin, and Stefanie Jegelka. A principled deep random field model for image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[78] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[79] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 28(10):1568–1583, 2006.

[80] Nikos Komodakis. Efficient training for pairwise or higher order crfs via dual decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[81] Nikos Komodakis and Nikos Paragios. Beyond pairwise energies: Efficient optimization for higher-order mrfs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[82] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. Mrf optimization via dual decomposition: Message-passing revisited. In *IEEE International Conference on Computer Vision (ICCV)*, 2007.

[83] Anoop Korattikara, Yutian Chen, and Max Welling. Austerity in mcmc land: Cutting the metropolis-hastings budget. In *International Conference on Machine Learning (ICML)*, 2014.

[84] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[85] Andreas Krause and Carlos Guestrin. `http://submodularity.org/`, 2016. [Online; accessed 2-August].

[86] Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[88] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[89] Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. In *Advances in Neural Information Processing Systems (NIPS)*, volume 20, pages 785–792, 2007.

[90] Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*, 2015.

[91] L'ubor Ladickỳ, Paul Sturgess, Chris Russell, Sunando Sengupta, Yalin Bastanlar, William Clocksin, and Philip HS Torr. Joint optimization for object class segmentation and dense stereo reconstruction. *International Journal of Computer Vision (IJCV)*, 100(2):122–133, 2012.

[92] L'ubor Ladickỳ, Chris Russell, Pushmeet Kohli, and Philip H.S. Torr. Inference methods for crfs with co-occurrence statistics. *International Journal of Computer Vision (IJCV)*, 103(2):213–225, 2013.

[93] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001.

[94] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2008.

[95] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

[96] Nicolas Le Roux, Nicolas Heess, Jamie Shotton, and John Winn. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011.

[97] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[98] Chi-Hoon Lee, Shaojun Wang, Feng Jiao, Dale Schuurmans, and Russell Greiner. Learning to model spatial dependency: Semi-supervised discriminative random fields. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[99] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning (ICML)*, 2009.

[100] Victor Lempitsky, Pushmeet Kohli, Carsten Rother, and Toby Sharp. Image segmentation with a bounding box prior. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[101] Yujia Li and Richard S. Zemel. High order regularization for semi-supervised learning of structured output problems. In *International Conference on Machine Learning (ICML)*, 2014.

[102] Yujia Li and Richard S. Zemel. Mean field networks. In *ICML workshop on Learning Tractable Probabilistic Models*, 2014.

[103] Yujia Li, Kevin Swersky, and Richard S. Zemel. Learning unbiased features. In *NIPS workshop on Transfer and Multi-Task Learning*, 2014.

[104] Yujia Li, Kevin Swersky, and Richard S. Zemel. Generative moment matching networks. In *International Conference on Machine Learning (ICML)*, 2015.

[105] Yujia Li, Kaisheng Yao, and Geoffrey Zweig. Feedback-based handwriting recognition from inertial sensor data for wearable devices. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[106] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[107] Percy Liang, Michael I. Jordan, and Dan Klein. Learning from measurements in exponential families. In *International Conference on Machine Learning (ICML)*, 2009.

[108] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[109] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[110] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard S. Zemel. The variational fair auto encoder. In *International Conference on Learning Representations (ICLR)*, 2016.

[111] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[112] Alessandro Lusci, Gianluca Pollastri, and Pierre Baldi. Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. *Journal of chemical information and modeling*, 53(7):1563–1575, 2013.

[113] Michael Maire, Pablo Arbeláez, Charless Fowlkes, and Jitendra Malik. Using contours to detect and localize junctions in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008.

[114] Gideon S. Mann and Andrew McCallum. Generalized expectation criteria for semi-supervised learning with weakly labeled data. *Journal of Machine Learning Research (JMLR)*, 2010.

[115] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[116] Roland Memisevic and Geoffrey E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, 2010.

[117] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[118] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[119] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

[120] Kevin Miller, M. Pawan Kumar, Benjamin Packer, Danny Goodman, Daphne Koller, et al. Max-margin min-entropy models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[121] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Conference on Uncertainty and Artificial Ingelligence (UAI)*, 2001.

[122] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning (ICML)*, 2014.

[123] Volodymyr Mnih, Hugo Larochelle, and Geoffrey E. Hinton. Conditional restricted boltzmann machines for structured output prediction. In *Conference on Uncertainty and Artificial Ingelligence (UAI)*, 2011.

[124] Radford M. Neal. Probabilistic inference using markov chain monte carlo methods. Technical report, Department of Computer Science, University of Toronto, 1993.

[125] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. Learning to classify text from labeled and unlabeled documents. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1998.

[126] Mohammad Norouzi, Mani Ranjbar, and Greg Mori. Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[127] Sebastian Nowozin and Christoph H. Lampert. Global connectivity potentials for random field models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[128] Sebastian Nowozin and Christoph H. Lampert. Structured learning and prediction in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

[129] Peter O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *International Workshop on Computer Science Logic (CSL)*, 2001.

[130] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[131] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[132] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229, 1987.

[133] Ruzica Piskac, Thomas Wies, and Damien Zufferey. GRASShopper - complete heap verification with mixed specifications. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2014.

[134] Renfrey Burnard Potts. Some generalized order-disorder transformations. In *Mathematical proceedings of the cambridge philosophical society*, volume 48, pages 106–109. Cambridge Univ Press, 1952.

[135] Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Michael Collins, and Trevor Darrell. Hidden conditional random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 29(10):1848–1852, 2007.

[136] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *IEEE Symposium on Logic in Computer Science (LICS)*, 2002.

[137] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 2004.

[138] Carsten Rother, Pushmeet Kohli, Wei Feng, and Jiaya Jia. Minimizing sparse higher order energy functions of discrete variables. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[139] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* MIT Press, 1985.

[140] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[141] Alexander Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Efficient structured prediction with latent variables for general graphical models. In *International Conference on Machine Learning (ICML)*, 2012.

[142] H.J. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371, 1965.

[143] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 2011.

[144] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2014.

[145] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel distributed processing*. 1986.

[146] Richard Socher, Cliff C. Lin, Chris Manning, and Andrew Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *International Conference on Machine Learning (ICML)*, 2011.

[147] David Sontag, Talya Meltzer, Amir Globerson, Tommi S. Jaakkola, and Yair Weiss. Tightening lp relaxations for map using message passing. In *Conference on Uncertainty and Artificial Ingelligence (UAI)*, 2008.

[148] David Sontag, Amir Globerson, and Tommi Jaakkola. Introduction to dual decomposition for inference. *Optimization for Machine Learning*, 2011.

[149] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[150] Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 725–733, 2011.

[151] Amarnag Subramanya, Slav Petrov, and Fernando Pereira. Efficient graph-based semi-supervised learning of structured tagging models. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2010.

[152] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[153] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[154] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[155] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[156] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey E. Hinton. Robust boltzmann machines for recognition and denoising. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[157] Daniel Tarlow and Richard S. Zemel. Structured output learning with high order loss functions. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[158] Daniel Tarlow, Inmar E. Givoni, and Richard S. Zemel. Hop-map: Efficient message passing with high order potentials. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[159] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[160] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *International Conference on Machine Learning (ICML)*, 2008.

[161] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*, 2004.

[162] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (IJCV)*, 6, 2005.

[163] Werner Uwents, Gabriele Monfardini, Hendrik Blockeel, Marco Gori, and Franco Scarselli. Neural networks for relational learning: an experimental comparison. *Machine Learning*, 82(3):315–349, 2011.

[164] Alexander Vezhnevets, Vittorio Ferrari, and Joachim M. Buhmann. Weakly supervised semantic segmentation with a multi-image model. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[165] Sara Vicente, Vladimir Kolmogorov, and Carsten Rother. Graph cut based image segmentation with connectivity priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[166] Sara Vicente, Vladimir Kolmogorov, and Carsten Rother. Joint optimization of segmentation and appearance models. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[167] Luke Vilnis, David Belanger, Daniel Sheldon, and Andrew McCallum. Bethe projections for non-local inference. *arXiv preprint arXiv:1503.01397*, 2015.

[168] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[169] Martin J. Wainwright. Estimating the wrong graphical model: Benefits in the computation-limited setting. *Journal of Machine Learning Research (JMLR)*, 7:1829–1859, 2006.

[170] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

[171] Martin J. Wainwright, Tommi S. Jaakkola, and Alan S. Willsky. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717, 2005.

[172] Jun Wang, Tony Jebara, and Shih-Fu Chang. Graph transduction via alternating minimization. In *International Conference on Machine Learning (ICML)*, 2008.

[173] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-UCSD Birds 200. Technical report, California Institute of Technology, 2010.

[174] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[175] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: a set of prerequisite toy tasks. In *International Conference on Learning Representations (ICLR)*, 2016.

[176] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[177] Oliver J. Woodford, Carsten Rother, and Vladimir Kolmogorov. A global perspective on map inference for low-level vision. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[178] Jimei Yang, Simon Safar, and Ming-Hsuan Yang. Max-margin boltzmann machines for object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[179] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[180] Chun-Nam John Yu and Thorsten Joachims. Learning structural svms with latent variables. In *International Conference on Machine Learning (ICML)*, 2009.

[181] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H.S. Torr. Conditional random fields as recurrent neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[182] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[183] Xiaojin Zhu. Semi-supervised learning literature survey. Technical report, Department of Computer Science, University of Wisconsin-Madison, 2005.

[184] Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. Semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning (ICML)*, 2003.

[185] Alexander Zien, Ulf Brefeld, and Tobias Scheffer. Transductive support vector machines for structured variables. In *International Conference on Machine Learning (ICML)*, 2007.