UNDERSTANDING CONSTRAINTS IN STRUCTURED PREDICTION PROBLEMS

by

Xingyuan Pan

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2019

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Xingyuan Pan**
has been approved by the following supervisory committee members:

Vivek Srikumar ,	Chair(s)	**12 Sep 2019** Date Approved
Srikumar Ramalingam ,	Member	**08 Jul 2019** Date Approved
Ellen Riloff ,	Member	**29 May 2019** Date Approved
Suresh Venkatasubramanian ,	Member	**29 May 2019** Date Approved
Kai-Wei Chang ,	Member	**03 Jun 2019** Date Approved

by **Ross Whitaker** , Chair/Dean of
the Department/College/School of **Computing**
and by **David B. Kieda** , Dean of The Graduate School.

ABSTRACT

Structured prediction is the machine learning task of predicting a structured output given an input. For these problems constraints among output variables play significant roles in both the learning and prediction phases. The goal of this dissertation is to improve a structured-prediction model's performance by understanding constraints in the problem. More specifically, I will focus on learning useful constraints from data and reduce inference time at the presence of complicated constraints.

I use neural networks to learn constraints from the training data of structured prediction problems. I frame the problem as that of training shallow rectifier networks to identify valid structures or substructures. The trained rectifier networks are not readily useful in structured prediction problems. To convert the trained networks to a form which can be directly used in various structured prediction inference algorithms, I derive a result on the expressiveness of rectifier networks and use it to convert the trained networks to systems of linear inequality constraints. I empirically verify the effectiveness of the learned constraints by performing experiments in the information extraction domain. I show that the learned linear inequality constraints are very effective in improving the prediction performance of an existing structured prediction model, and they can even be used in the training phase to obtain a better classifier.

When complicated constraints exist, inference in a structured prediction problem is often hard. I consider the problem of reducing the inference time of a trained black-box classifier without losing accuracy. To do so, I train a speedup classifier that learns to imitate a black-box classifier under the learning to search approach. As the structured classifier predicts more examples, the speedup classifier will operate as a learned heuristic to guide search to favorable regions of the output space. Evaluations on the task of entity and relation extraction show that the speedup classifier outperforms even greedy search in terms of speed without loss of accuracy of the original black-box classifier.

In the last part of the dissertation, I derive a way of constructing the decision boundary

for deep neural networks with piecewise linear activations. This construction is an extension of the expressiveness results on shallow rectifier networks. The decision boundaries of such networks are composed of a system of hyperplanes. I identify these decision hyperplanes and describe a way of combining them to construct the decision boundaries of the deep neural networks.

For my parents, Xiaoyan Chen and Zexiang Pan.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Vivek Srikumar, for all the mentoring and support during my entire Ph.D. period. Vivek opened the door to machine learning for me and made me realize that this is an important and fascinating area to work on. I enjoyed every minute of discussions with him along the journey.

I want to thank Suresh Venkatasubramanian and Aditya Bhaskara for teaching me how to do theoretical research in computer science in general, and various kinds of algorithms in particular. The theory part of my dissertation would not be possible without their guidance. I thank Srikumar Ramalingam for many helpful discussions, especially for the qualifying exam question he gave me, which helped me derive a general result on deep neural networks. I want to thank Ellen Riloff for introducing me to the field of natural language processing, and for many pieces of advice on choosing research topics, academic writing and presentations, and on how to succeed as a Ph.D. student. I would also like to thank Kai-Wei Chang, who provided valuable suggestions on my qualifying exam.

I want to thank many of my friends from whom I learned a lot about different aspects of computer science research and who made my life much more enjoyable while pursuing my Ph.D. degree. They are Jie Cao, Rui Dai, Haibo Ding, Min Du, Yang Gao, Vivek Gupta, Tianyu Jiang, Cai Li, Tao Li, Qingkai Lu, Ashequl Qadir, Pingfan Tang, Xu Wang, Zhenduo Wang, Kui Wu, Xin Yu, Liang Zhang, Guineng Zheng, Yan Zheng, Yichu Zhou, and Yuan Zhuang.

Finally, I would like to thank my parents for their love and support during the long journey of my Ph.D. career. This dissertation is dedicated to you.

CHAPTER 1

INTRODUCTION

My dissertation is about structured prediction problems, especially as they appear in natural language processing (NLP) [94]. Applications of structured prediction problems in the NLP domain include dependency parsing, semantic role labeling, and various kinds of sequence labeling tasks such as part-of-speech tagging, named entity recognition, etc. Unlike standard classification or regression problems in which case we want to predict a single class label or a real number, a structured prediction problem requires predicting a structured output, i.e., multiple interdependent output variables in a coherent way. The interactions among output variables can be characterized by a probabilistic graphical model such as a factor graph with features [49], or expressed using constraints, usually taking the form of linear inequalities over output variables [85], or combinations of the two.

This dissertation specifically focuses on understanding the constraints in structured prediction problems from two perspectives. First, it is well known that constraints are essential in modeling a structured prediction task, and properly designed constraints can dramatically improve the model performance. However, designing constraints is not an easy task and often requires domain expertise in the area. I design a general framework of automatically discovering constraints as linear inequalities using the theory I developed on the expressiveness of neural networks. Second, in the presence of complicated constraints, the inference task, or predicting the output structure given a trained model, is often computationally hard since the number of possible outputs is usually exponentially large. I propose a way of learning a speedup classifier to imitate an already trained structured classifier. The speedup classifier can achieve much faster inference without losing accuracy of the original structured classifier.

In this introductory chapter, I give some concrete examples of the structured prediction problems along with the accompanying challenges. Then I briefly summarize the contribu-

tions of this dissertation, which can help to solve these challenging problems. The chapter ends with a road map for the rest of the dissertation.

1.1 Examples of structured prediction problems

In this section, I provide two examples of structured prediction problems, both information extraction tasks from the NLP domain. The first one is a sequence prediction task, and the goal is to extract citation fields from a given bibliography entry. The second one is a task in which we try to extract entities and relations among them from a given piece of text. Both examples are typical structured prediction problems that illustrate the characteristics of such problems. As I introduce these examples, I will point out some of the challenges associated with structured prediction problems, and my dissertation provides possible ways to tackle these challenges.

1.1.1 Sequence labeling: citation field extraction

An important kind of structured prediction problem is the sequence labeling problem. The input to a sequence labeling problem is a sequence, i.e., a list of tokens, usually words in NLP. We want to label each token with a tag from a set of predetermined tags.

A prototypical example of the sequence labeling problem might be the part-of-speech tagging problem, where we want to assign a part-of-speech tag such as noun, verb, or adjective to each word in a sentence. Some NLP tasks, which do not seem sequence labeling problems at first glance, can be framed as such problems and solved using a technique called IOB encoding [79]. Example problems of this kind include shallow parsing [92] and semantic role labeling etc [39].

In this subsection, I give a concrete example of sequence labeling problem, the citation field extraction problem. This example will introduce some of the common properties of structured prediction problems and the difficulties in solving them.

In the citation field extraction task, we are given a piece of text such as the following:

> A . M . Turing . Computing machinery and intelligence . Mind , 59 , 433-460 . October , 1950 .

This kind of bibliography entry often appears in the reference section of an academic publication. The goal of this task is to extract citation fields such as authors, title, dates, etc., as shown below:

[Author A . M . Turing .] [Title Computing machinery and intelligence .] [Journal Mind ,] [Volume 59 ,] [Pages 433-460 .] [Date October , 1950 .]

We can think of the citation field extraction problem as a sequence labeling problem by given every token in the sequence a label, and the interested citation fields can be recovered from the token labels. More specifically, we label every token that begins a span of an interested field with label B-<field name>, tokens that inside a field with label I-<field name>. For instance, the label for the first token in the above example is B-Author; the labels for the next several tokens are all I-Author, until we reach the word "Computing", where the label is B-Title.

It is unwise to assign the citation field tags individually for each token in the bibliography entry; rather, we want a joint assignment to all the tokens so that the whole assignment forms a coherent tag sequence. The most often used sequence models are the first order Markov models, which include hidden Markov model (HMM), maximum entropy Markov model (MEMM), sequence condition random field (CRF), etc. Such models capture pairwise interactions between adjacent token labels, and prohibit, for example, an I-Title following a B-Author. However, a first-order sequence model does not capture long-range interactions and global constraints. Suppose we want to capture the following two constraints in our model:

1. Each citation must contain either a TITLE or a BOOKTITLE, but not both.

2. If a citation contains the field BOOKTITLE, it must contain the field PUBLISHER.

These are just two example constraints that are global in nature and cannot be captured in the first order sequence model. In fact, they cannot be captured by any sequence model based on Markov assumptions. Yet, constraints like these have proved to be important for successfully modeling this task [1, 8, 9]. We can imagine that there are many other constraints that can be useful. Therefore, we want a systematic way of automatically discovering constraints from data rather than manually designing them, which is a tedious and time-consuming process requiring domain expertise.

1.1.2 Entity and relation extraction

In the task of entity and relation extraction, we are given a piece of text (a sentence or multiple sentences) such as:

Colin went back home in Ordon Village.

The goal is to determine the class label of each entity (underlined in the above example), and the relation between each pair of entities. We assume the entity candidates are given either from human annotations or from a preprocessing step. Let us consider three types of entities, `person`, `location` and `organization`, and five types of relations, `Kill`, `LiveIn`, `WorkFor`, `LocatedAt` and `OrgBasedIn`. Additionally, there is a special entity label `NoEnt` meaning a text span is not an entity, and a special relation label `NoRel` indicating that two spans are unrelated. The target output structure is shown in Fig. 1.1.

It is easy to see that we do not want to determine each entity and relation independently since there are many natural interactions between labels in this simple domain. Two example constraints might be

1. If entity A is a `person`, and entity B is a `location`, the relation from entity A to entity B must be either `LiveIn` or `NoRel`.

2. If an entity is labeled as `NoEnt`, then any relations involving it must be `NoRel`.

Constraints like these are easy to understand. The questions I want to ask (and answer) are whether we can discover such constraints, or perhaps more profound ones that are not so easy to describe, automatically from the data, and how we can use the constraints once we discovered them.

Another kind of challenge arises once the size of the input instance gets large. In the above example, we just have two candidate spans, and only four decisions need to be made. It is not uncommon to have a situation in which there are many dozens of candidate spans in a piece of text. In this case, the inference process is computationally hard because of the combinatorially large output space that one needs to search. In fact, the worst

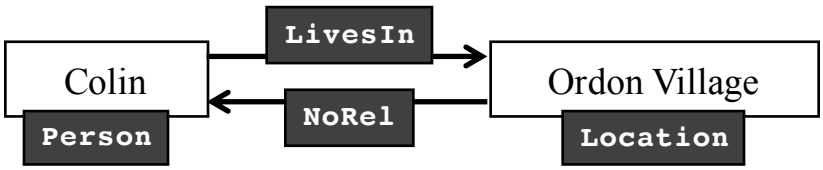

Figure 1.1. An example of the entities and relations task. The nodes are entity candidates, and directed edges indicate relations. The labels in typewriter font are the decisions that we need to make.

case intractability results of MAP inference is well studied [45, 72]. In this dissertation, I propose a general way of speeding up the inference process of a given structured classifier by training a search-based classifier that imitates the given classifier.

1.2 Contributions of this dissertation

From the discussions in Section 1.1, one can see that we need a systematic approach to learn constraints. We know that recent advances in neural network research have made great progress in solving the feature representation problem. It is natural to ask if it is possible to use neural networks to discover constraints for structured prediction problems. Furthermore, can we treat the problem of predicting constrained structured outputs as a learnable concept? This dissertation proposes the following ideas:

- It is possible and indeed beneficial to use neural networks to learn constraints from data for structured prediction problems.
- A given structural classifier can learn to predict fast as it predicts more and more examples while maintaining its accuracy.

The primary contributions of the dissertation can be summarized with the following three aspects:

1. I analyze the expressiveness of neural networks with certain piecewise linear activation functions. When a neural network of this kind is used as a classification function, I show that it can be converted into a system of linear inequalities over the input to the network. This result not only provides a formal analysis of the expressiveness of the neural network but also serves as a basis for learning constraints using neural networks.

2. I design a general framework for discovering linear constraints from labeled data for structured prediction problems. I frame the problem as that of training a shallow rectifier network to identify valid structures or substructures, and convert the resulting network to linear inequality constraints using the construction I proposed in developing the expressiveness results on neural networks.

3. I provide a formalization of the problem of learning to make structured output classifiers faster without sacrificing accuracy. I develop a learning-to-search approach to train a speedup classifier with a mistake bound guarantee and a sufficient con-

dition to safely avoid computing input-based features. I show empirically on an entity-relation extraction task that the speedup classifier is faster than both the state-of-the-art Gurobi optimizer and greedy search, without incurring a loss in output quality.

1.3 Road map of this dissertation

The rest of this dissertation is organized as follows:

- Chapter 2 introduces the background material on the expressiveness of neural networks, as well as structured prediction problems. Previous related work is also discussed in this chapter.

- Chapter 3 introduces my theoretical results on the expressiveness of shallow rectifier networks. Comparison is made to the expressiveness of shallow threshold networks. I provide a theory that can be used to convert a shallow rectifier network into an equivalent shallow threshold network. This result is used in Chapter 4 to discover constraints from data.

- Chapter 4 introduces a general framework for learning constraints from the training data of the structured prediction problems. The theoretical foundation is based on the expressiveness results of neural networks introduced in Chapter 3. Empirical verifications demonstrate the effectiveness of the proposed constraint learning framework.

- Chapter 5 provides a way of speeding up the inference process of a given structured prediction classifier. The learning to search approach is introduced in this chapter, which is the base of our speedup classifier. The speedup heuristic function is defined, and it is shown empirically that the speedup classifier based on an appropriately learned heuristic can dramatically reduce the inference time without losing accuracy. A mistake bound theorem is also given to identify the conditions under which the speedup learning algorithm will converge.

- Chapter 6 extends the previous results on shallow rectifier networks in Chapter 2 to the deep neural network case. This chapter also discusses networks with activation

functions other than rectifiers. I provide a general approach for constructing decision boundaries of deep neural networks with piecewise linear activation functions. The results from this chapter help to explain theoretically why deep neural networks are so successful in practice.

- Chapter 7 concludes the dissertation and discusses possible future research directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, I introduce the background of this dissertation, as well as previous related work. Since I will use neural networks to learn constraints from data for structured prediction problems, I will start by introducing the expressiveness of neural networks in Section 2.1. It is beyond the scope of this dissertation to thoroughly introduce all the theoretical results we know about neural networks; therefore, I only provide relevant previous results on the expressiveness of neural networks, with pointers for the interested reader. Section 2.2 formally defines the structured prediction problems, with focus given to the roles played by the constraints in these problems.

2.1 Expressiveness of neural networks

A feedforward neural network specifies a function $F : \mathbb{R}^{n_0} \to \mathbb{R}^{n_y}$ that is given by the following form:

$$\begin{aligned} \mathbf{y}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \mathbf{h}^{(l)} &= g(\mathbf{y}^{(l)}), \quad l \in [L] \\ \mathbf{y}^{(l+1)} &= \mathbf{W}^{(l+1)}\mathbf{h}^{(l)} + \mathbf{b}^{(l+1)}, \quad l \in [L]. \end{aligned} \tag{2.1}$$

This is an L-hidden-layer network, in which each layer receives previous layer's output as input. We assume that the input dimension is n_0, that is $\mathbf{x} \in \mathbb{R}^{n_0}$, and the number of units in the l^{th} layer is n_l for $l \in [L]$. The output layer has n_y units, and the output of the network is $\mathbf{y} = \mathbf{y}^{L+1} \in \mathbb{R}^{n_y}$. The parameters of the network are $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$. The *nonlinear* activation function in the hidden layers are specified by g. Notice that the output layer is a linear layer.

In this dissertation, we are interested in the cases of using neural network for classification problems, in particular, binary classification problems. Therefore, we only consider the cases in which the network has only one output unit. We are interested in the con-

ditions under which the output $y \geq 0$. Equivalently, we are interested in specifying the decision boundary separating the input space $\mathbb{R}^{n_0}$. All inputs $\mathbf{x}$ at one side of the decision boundary will result in a positive output y, and all inputs $\mathbf{x}$ at the other side of the decision boundary will result in a negative y. The construction of the decision boundary of the shallow rectifier network is the topic of Chapter 3.

A neural network, when used as a function for classification, can be used in a structured prediction setting to identify valid structures or substructures. If we train a neural network for this purpose, we can use it in the inference or training process of a structured prediction problem. This will be discussed extensively in Chapter 4.

2.1.1 Related work

A neural network is characterized by its architecture, the choices of activation functions, and its parameters. We see several activation functions in the literature – the most common ones being the threshold, logistic, hyperbolic tangent and rectified linear units (ReLUs). In recent years, deep neural networks with rectifying neurons – defined as $\text{rect}(x) = \max(0, x)$ – have shown state-of-the-art performance in several tasks such as image and speech classification [30, 47, 55, 67, 112].

ReLUs possess several attractive computational properties. First, compared to deep networks with sigmoidal activation units, ReLU networks are less affected by the vanishing gradient problem [2, 30, 41]. Second, rectifying neurons encourage sparsity in the hidden layers [30]. Third, gradient backpropagation is efficient because of the piece-wise linear nature of the function. For example, Krizhevsky et al. [47] report that a convolutional neural network with ReLUs is six times faster than an equivalent one with hyperbolic tangent neurons. Finally, they have been empirically shown to generalize very well. Despite these clear computational and empirical advantages, the expressiveness of rectifier units is less studied, unlike sigmoid and threshold units.

From the learning point of view, the choice of an activation function is driven by two related aspects: the expressiveness of a given network using the activation function, and the computational complexity of learning. Though this work studies the former, we briefly summarize prior work along both these lines.

Any continuous function can be approximated to arbitrary accuracy with only one

hidden layer of sigmoid units [16], leading to neural networks being called "universal approximators." With two layers, even discontinuous functions can be represented. Moreover, the approximation error (for real-valued outputs) is insensitive to the choice of activation functions from among several commonly used ones [19], provided we allow the size of the network to increase polynomially and the number of layers to increase by a constant factor. Similarly, two-layer threshold networks are capable of representing any Boolean function. However, these are existence statements; for a general target function, the number of hidden units may be exponential in the input dimensionality. Maass et al. [56, 57] compare sigmoid networks with threshold networks and point out that the former can be more expressive than similar-sized threshold networks.

There has been some recent work that looks at the expressiveness of feed-forward ReLU networks. Because the rectifier function is piece-wise linear, any network using only ReLUs can only represent piece-wise linear functions. Thus, the number of linear partitions of input space by the network can be viewed as a measure of its expressiveness. Pascanu et al. [64, 73] show that for the same number of ReLUs, a deep architecture can represent functions with exponentially more linear regions than a shallow architecture. While more linear regions indicate that more complex functions can be represented, it does not directly tell us how expressive a function is; at prediction time, we cannot directly correlate the number of regions to the way we make the prediction. Another way of measuring the expressiveness of feed-forward networks is by considering its classification error; Telgarsky [102] compares shallow and deep ReLU networks in this manner.

The learning complexity of neural networks using various activation functions has also been studied. For inputs from the Boolean hypercube, the two-layer networks with threshold activation functions is not efficiently learnable [4, 17, 44]. Without restricting the weights, two-layer networks with sigmoid or ReLU activations are also not efficiently learnable. We also refer the reader to the work by Livni et al. [53] that summarizes and describes positive and negative learnability results for various activations.

2.2 Structured prediction

Structured prediction is the machine learning task of predicting an output structure rather than a class label. The outputs can take the form of sequences, trees, or in general,

labeled graphs. Many natural language processing (NLP) tasks can naturally be framed as structured prediction [94].

Formally, in a structured prediction problem, we are given a structured input $\mathbf{x}$ and a corresponding feasible set of possible output structures $\mathcal{Y}_{\mathbf{x}}$, and the goal is to produce a structured output $\mathbf{y}^* \in \mathcal{Y}_{\mathbf{x}}$ by maximizing a linear scoring function,

$$\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} \alpha \cdot \Phi(\mathbf{x}, \mathbf{y}), \tag{2.2}$$

where $\Phi(\mathbf{x}, \mathbf{y})$ is a feature vector representation for the input-output pair $(\mathbf{x}, \mathbf{y})$, and α is a weight vector learned from training data. The feature representation Φ can be designed by hand using domain knowledge, or learned using deep neural networks [31, 32]. In this dissertation, we always assume that the feature representation is given to us.

An output structure $\mathbf{y}$ can be determined from a set of discrete variables $\{y_1, y_2, \cdots\}$, or equivalently, the structure $\mathbf{y}$ is a deterministic function of the discrete variables $\{y_1, y_2, \cdots\}$. Seeing this way, the structure prediction problem is a combinatorial optimization problem.

In structured prediction, there are usually interdependencies among output variables. From a practical point of view, the interactions among variables manifest themselves in three different ways: (i) Multiple variables may be involved in a factor in an underlying graphical model; therefore, the corresponding feature function depends on these variables simultaneously. (ii) Common variables in neighboring factors must have consistent assignments to form a valid output. (iii) Domain knowledge may dictate some additional constraints among output variables. Note that points (i) and (ii) are the minimum requirements for modeling any nontrivial structured prediction problem. In contrast, domain knowledge constraints are not an absolute must; in principle, we can still have a valid structured prediction problem without them. However, it has been shown that carefully designed constraints can substantially improve the model performance in various NLP applications [1, 9, 85].

It is unwise to predict each variable y_i independently since the resulting structure might not be feasible, as defined by the membership to the set $\mathcal{Y}_{\mathbf{x}}$ in Eq. (2.2). Rather, there are constraints among these output variabels y_i's. The constraints usually take the form of linear inequalities of output variables, and they can be either designed by using domain knowledge of the problem or learned from the data.

Solving Eq. (2.2) to obtain the best structure is called the *inference* problem, and it is generally hard due to the combinatorially large output space $\mathcal{Y}_{\mathbf{x}}$ [45, 72]. One common way to solve inference is by designing efficient dynamic programming algorithms that exploit problem structure. While effective, this approach is limited to special cases where the problem admits efficient decoding, thus placing restrictions on factorization and feature design. In the worst case, the inference problem is computationally intractable due to the combinatorially large output space $\mathcal{Y}_{\mathbf{x}}$, regardless of how the inference problem is formulated. For example, if the inference problem is formulated as a MAP query in a graphical model, we know that MAP inference is NP-hard [45]. As another example, if the inference problem is formulated as an integer linear program (ILP), we know that ILP is also NP-hard [90].

In my dissertation, I seek to reason about the problem of predicting structures in the general case. Since inference is essentially a combinatorial optimization problem, without loss of generality, we can represent any inference problem as an integer linear programming (ILP) instance [90]. For a given input example $\mathbf{x}$, the inference task in Eq. (2.2) can be converted into an ILP instance,

$$\min_{\mathbf{z}} \sum_i c_i z_i \tag{2.3}$$

$$\mathbf{A}\mathbf{z} \geq \mathbf{b} \tag{2.4}$$

$$z_i \in \{0, 1\}, \tag{2.5}$$

where the indicator variables z_i's are constructed by systematic binarization[1] of the discrete decisions y_i, and $\mathbf{z}$ is a column vector representation of all the indicator variables z_i. The coefficients c_i's will depend on the input example and the weight α. $\mathbf{A}$ is a constraint matrix and $\mathbf{b}$ is a constraint vector, and Eq. (2.4) is an element-wise inequality. For illustration of writing inference of structured prediction as an ILP instance please refer to the work of Roth and Yih [85].

Note that it is those constraints in Eq. (2.4) that make the inference in structured prediction computationally hard since they couple together different indicator variables z_i. My dissertation will focus on discovering constraints from data rather than manually

[1] For example, each z_i component could denote an indicator for a decision.

designing them from domain knowledge (Chapter 4), and on improving the inference speed at the presence of complicated constraints (Chapter 5). The learned constraints are complementary with the hand designed constraints and can be combined with the hand-designed ones to give a more accurate structure classifier or a faster structure classifier. More specifically, I will consider the following two separate scenarios when learning constraints for structured prediction problems:

1. Learning constraints from labeled structures. During the training process, we have a set of labeled structured prediction examples. We could use these training examples with any learning algorithm we choose to learn a structured classifier. However, additionally, we would like to learn constraints from the labeled structures. The goal is to use the learned constraints to train an improved model, which can be subsequently used during test time.

2. Learning constraints from unlabeled data. In this scenario, we are given an already trained structure classifier. This classifier is used to predict previously unseen examples, which are unlabeled. We want to take the opportunity of predicting new examples to learn regularities in constrained prediction. The learned constraints can help speed up future predictions.

2.2.1 Related work

Many natural language processing (NLP) tasks naturally lend themselves as structured prediction problems. Examples include constituent and dependency parsing [14, 24, 61, 81, 93], semantic role labeling [74, 82, 86], various kinds of sequence labeling [13, 49, 62, 92, 95], and information extraction [1, 26, 58, 85] tasks. For a survey of the application of structured prediction to NLP tasks, see the textbook by Smith [94].

It has been shown that carefully designed constraints can substantially improve the model performance in various kinds of NLP applications [1, 9, 75, 85]. The constraints can be hand-designed from domain expertise or learned from data. Regardless of where the constraints come from, we need a way of combining the constraints with the structured prediction model to make predictions on new examples of a structured prediction problem.

The exact way of incorporating the constraints depends on the inference algorithm used by the original model. For example, if a structured prediction problem is formu-

lated as an integer linear program (ILP) [48, 61, 81, 85, 86, 98], the learned constraints will become additional linear inequalities, which can be easily incorporated into the ILP formalism. Alternatively a structure can be constructed using graph search [3, 7, 14, 20, 21, 23, 42, 52, 54, 71, 84, 105, 109]. The state in the search space usually corresponds to some partial assignment of the output variables. The successor states then correspond to the possible assignment of the unassigned variables. The role of the learned constraints is to filter the available actions during search-node expansion. Other inference techniques such as Lagrangian Relaxation [10, 18, 46, 87, 88, 96] can also employ the learned constraints. These techniques require that we have a base predictor which can efficiently solve the unconstrained version of the same problem. One then repeatedly updates the Lagrangian multiplier to obtain the optimal solution to the original problem.

Constraint learning, or more general, rule learning, has many forms and has been studied in various subfields of artificial intelligence. Quinlan [77] describes the ID3 induction algorithm, which extracts rules in the form of a decision tree. SLIPPER [12] is an ensemble based rule learning system, which repeatedly boosts a weak rule-builder.

First order logic rules can be learned from examples using inductive logic programming [50, 66, 69]. Notable example systems using inductive logic programming include FOIL [76] and Progol [65], among others. Constraint satisfaction problems (CSPs) have been studies extensively in constraint programming community [78].

Statistical relation learning deals with learning constraints with uncertainty. Markov logic network [80], an undirected relational model, is a framework that combines probabilistic modeling with external knowledge in terms of first-order logic. The logic formula can be viewed as soft constraints, and the weights can be learned from data. Wang et al. [107] proposed a structure-learning method for the probabilistic logic ProPPR [106, 108], and showed that first-order logic rules could be generated by parameter learning. Structure learning in directed probabilistic logic models has also been explored [27–29].

Constraint learning is also related to structure learning in probabilistic graphical models. In both directed and undirected graphical models, the graphical structure specifies the independence relations, and certain independence relations can be expressed as constraints over the possible values of the random variables. For example, in Bayesian networks, the conditional probability distributions may prohibit certain values at some nodes,

which lead to hard constraints for the problem. In conditional random field, a hard constraint can be viewed as a factor with infinite potential.

The dominant way of learning graphical structures is to assign scores to possible structures in the hypothesis space, and then search for the structure which maximizes the score function. The scoring function is usually some taken to be the likelihood function. This approach of structure learning was pioneered by the work of [11] on learning three-structured Bayesian networks. Another special case of learning Bayesian networks is when the order of nodes is predetermined. The $K2$ algorithm proposed by Cooper and Herskovits [15] can efficiently learn the graphical structure in this case. Learning general structures remains a computationally hard problem due to the combinatorially large search space of all possible structures. The textbook by Koller and Friedman [45] provides a thorough treatment of the topic of learning graphical models.

CHAPTER 3

EXPRESSIVENESS OF SHALLOW RECTIFIER NETWORKS

Rectified Linear Units (ReLUs) have been shown to ameliorate the vanishing gradient problem, allow for efficient backpropagation, and empirically promote sparsity in the learned parameters. They have led to state-of-the-art results in a variety of applications. However, unlike threshold and sigmoid networks, ReLU networks are less explored from the perspective of their expressiveness. This chapter studies the expressiveness of ReLU networks. We characterize the decision boundary of two-layer ReLU networks by constructing functionally equivalent threshold networks. We show that while the decision boundary of a two-layer ReLU network can be captured by a threshold network, the latter may require an exponentially larger number of hidden units. We also formulate sufficient conditions for a corresponding logarithmic reduction in the number of hidden units to represent a sign network as a ReLU network. Finally, we experimentally compare threshold networks and their much smaller ReLU counterparts with respect to their ability to learn from synthetically generated data. The results in this chapter will be used as a theoretical foundation of learning constraints for structured prediction problems.

More specifically, we address the following question: *Which Boolean functions do ReLU networks express?* We analyze the expressiveness of shallow ReLU networks by characterizing their equivalent threshold networks. The goal of our analysis is to offer a formal understanding of the successes of ReLUs by comparing them to well-studied threshold functions [36]. To this end, the contributions of this chapter are the following:

1. We provide a constructive proof that two-layer ReLU networks are equivalent to exponentially larger threshold networks. Furthermore, we show that there exist two-layer ReLU networks that *cannot* be represented by any smaller threshold networks.

2. We use this characterization to define a sufficient condition for compressing an arbi-

trary threshold network into a logarithmically smaller ReLU network.

3. We identify a relaxation of this condition that is applicable if we treat hidden layer predictions as a multiclass classification, thus requiring equivalence of hidden layer states instead of the output state.

3.1 What do ReLUs express?

Recall that we primarily focus on shallow networks with one hidden layer with n units and a single binary output. In all cases, the hidden layer neurons are the object of study. The output activation function is always the threshold function. In the rest of the dissertation, I use boldfaced letters to denote vectors. Input feature vectors and output binary labels are represented by $\mathbf{x}$ and $y \in \{\pm 1\}$, respectively. The number of hidden units is n. The weights and bias for the k^{th} rectifier are $\mathbf{u}_k$ and b_k; the weights and bias for the k^{th} sign units are $\mathbf{v}_k$ and d_k. The weights for the output unit are w_1 through w_n, and its the bias is w_0.

3.1.1 Threshold networks

Before coming to the main results, we will first review the expressiveness of threshold networks. Assuming there are n hidden units and one output unit, the output of the network can be written as

$$y = \operatorname{sgn}\left(w_0 + \sum_{k=1}^{n} w_k \operatorname{sgn}\left(\mathbf{v}_k \cdot \mathbf{x} + d_k\right)\right). \tag{3.1}$$

Here, both hidden and output activations are the sign function, which is defined as $\operatorname{sgn}(x) = 1$ if $x \geq 0$, and -1 otherwise. Each hidden unit represents one hyperplane (parameterized by $\mathbf{v}_k$ and d_k) that bisects the input space into two half spaces. By choosing different weights in the hidden layer, we can obtain arbitrary arrangement of n hyperplanes. The theory of hyperplane arrangement [111] tells us that for a general arrangement of n hyperplanes in d dimensions, the space is divided into $\sum_{s=0}^{d} \binom{n}{s}$ regions. The output unit computes a linear combination of the hidden output (using the w's) and thresholds it. Thus, for various values of the w's, threshold networks can express intersections and unions of those regions. Figure 3.1 shows an example of the decision boundary of a two-layer network with three threshold units in the hidden layer.

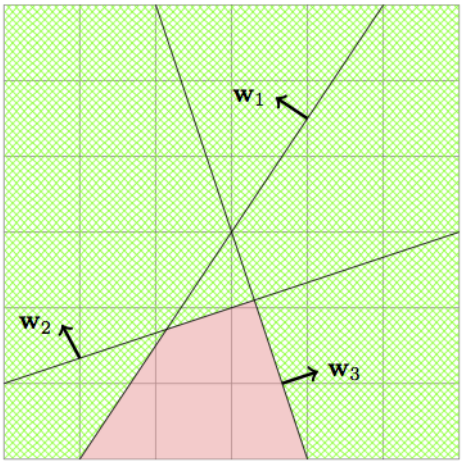

Figure 3.1. An example of the decision boundary of a two-layer network in two dimensions, with three threshold units in the hidden layer. The arrows point towards the half-space that is classified as positive (the green checked region).

3.1.2 Rectifier networks

In this subsection, we will show that the decision boundary of every two-layer neural network with rectifier activations can be represented using a network with threshold activations with two or three layers. However, the number of hidden threshold units can be exponential compared to the number of hidden rectifier units. We begin by defining the *rectifier activation function*.

Definition 3.1. *The rectifier activation function* $\text{rect}(\cdot)$ *is defined as*

$$\text{rect}(s) = \max\{0, s\}$$

for any real number $s \in \mathbb{R}$.

Even though the rectifier function $\text{rect}(\cdot)$ is defined on real numbers, we extend the definition so that the rectifier function can also be applied element-wise to a vector $\mathbf{z} \in \mathbb{R}^n$ to obtain $\text{rect}(\mathbf{z}) \in \mathbb{R}^n$, where the i^{th} component of $\text{rect}(\mathbf{z})$ is $\max\{0, z_i\}$.

Consider a network with one hidden layer of n ReLUs. For a d dimensional input $\mathbf{x}$, the output y is computed as

$$y = \text{sgn}\left(w_0 + \sum_{k=1}^{n} w_k \,\text{rect}(\mathbf{u}_k \cdot \mathbf{x} + b_k)\right). \tag{3.2}$$

Here $\mathbf{u}_k$ and b_k are weight and bias parameters for the ReLUs in the hidden layer, and the w_k's parameterize the output unit. To simplify notation, we will use a_k to denote the the preactivation input of the k^{th} hidden unit. That is, $a_k(\mathbf{x}) = \mathbf{u}_k \cdot \mathbf{x} + b_k$. This allows us to simplify the output as $\operatorname{sgn}\left(w_0 + \sum_{k\in[n]} w_k \operatorname{rect}(a_k(\mathbf{x}))\right)$. Here, $[n]$ is the set of positive integers not more than n. Note that even when not explicitly mentioned, each a_k depends on the $\mathbf{u}_k$ and the b_k parameters.

By definition of the rectifier, for any real number c, we have $c \operatorname{rect}(x) = \operatorname{sgn}(c) \operatorname{rect}(|c|x)$. Thus, we can absorb $|w_k|$ into the rectifier function in Eq. (3.2) without losing generality. That is, other than w_0, all the other output layer weights are only relevant up to sign because their magnitude can be absorbed into hidden layer weights. We can partition the hidden units into two sets $\mathcal{P}$ and $\mathcal{N}$, depending on the sign of the corresponding w_k. That is, let $\mathcal{P} = \{k : k \in [n] \text{ and } w_k = +1\}$ and let $\mathcal{N} = \{k : k \in [n] \text{ and } w_k = -1\}$. We will refer to these partitions as the *positive* and *negative* hidden units, respectively.

This observation lets us state the general form of two-layer ReLU networks as

$$y = \operatorname{sgn}\left(w_0 + \sum_{k\in\mathcal{P}} \operatorname{rect}(a_k(\mathbf{x})) - \sum_{k\in\mathcal{N}} \operatorname{rect}(a_k(\mathbf{x}))\right). \tag{3.3}$$

The following two layer rectifier network will serve as our running example through out the chapter:

$$y = \operatorname{sgn}\left(w_0 + \operatorname{rect}(a_1(\mathbf{x})) - \operatorname{rect}(a_2(\mathbf{x})) - \operatorname{rect}(a_3(\mathbf{x}))\right). \tag{3.4}$$

This network consists of three ReLUs in the hidden layer, one of which positively affects the preactivation output, and the other two decrease it. Hence, the set $\mathcal{P} = \{1\}$ and the set $\mathcal{N} = \{2, 3\}$.

Using the general representation of a two-layer network with rectifier hidden units (Eq. (3.3)), we can now state our main theorem that analyzes the decision boundary of rectifier networks.

Theorem 3.1 (Expressiveness Theorem [70])**.** *Consider a two-layer rectifier network with n hidden units represented in its general form (Eq.* (3.3)*). Then, for any input* $\mathbf{x}$*, the following conditions are equivalent:*

1. *The network classifies the example* $\mathbf{x}$ *as positive.*

2. *There exists a subset $\mathcal{S}_1$ of $\mathcal{P}$ such that, for every subset $\mathcal{S}_2$ of $\mathcal{N}$, we have $w_0 + \sum_{k\in\mathcal{S}_1} a_k(\mathbf{x}) - \sum_{k\in\mathcal{S}_2} a_k(\mathbf{x}) \geq 0$.*

3. *For every subset $\mathcal{S}_2$ of $\mathcal{N}$, there exists a subset $\mathcal{S}_1$ of $\mathcal{P}$ such that $w_0 + \sum_{k\in\mathcal{S}_1} a_k(\mathbf{x}) - \sum_{k\in\mathcal{S}_2} a_k(\mathbf{x}) \geq 0$.*

Before discussing the implications of the theorem, let us see how it applies to our running example in Eq. (3.4). In this example, $\mathcal{P}$ has two subsets: $\emptyset$ and $\{1\}$, and $\mathcal{N}$ has four subsets: $\emptyset$, $\{2\}$, $\{3\}$ and $\{2,3\}$. The first and second conditions of Theorem 3.1 indicate that the prediction is positive if, and only if, the system of inequalities in (3.5) hold, or the system of inequalities in (3.6) hold.

$$\begin{cases} w_0 \geq 0, & (\text{with } \mathcal{S}_1 = \emptyset, \mathcal{S}_2 = \emptyset) \\ w_0 - a_2(\mathbf{x}) \geq 0, & (\text{with } \mathcal{S}_1 = \emptyset, \mathcal{S}_2 = \{2\}) \\ w_0 - a_3(\mathbf{x}) \geq 0, & (\text{with } \mathcal{S}_1 = \emptyset, \mathcal{S}_2 = \{3\}) \\ w_0 - a_2(\mathbf{x}) - a_3(\mathbf{x}) \geq 0. & (\text{with } \mathcal{S}_1 = \emptyset, \mathcal{S}_2 = \{2,3\}) \end{cases} \tag{3.5}$$

or

$$\begin{cases} w_0 + a_1(\mathbf{x}) \geq 0, & (\text{with } \mathcal{S}_1 = \{1\}, \mathcal{S}_2 = \emptyset) \\ w_0 + a_1(\mathbf{x}) - a_2(\mathbf{x}) \geq 0, & (\text{with } \mathcal{S}_1 = \{1\}, \mathcal{S}_2 = \{2\}) \\ w_0 + a_1(\mathbf{x}) - a_3(\mathbf{x}) \geq 0, & (\text{with } \mathcal{S}_1 = \{1\}, \mathcal{S}_2 = \{3\}) \\ w_0 + a_1(\mathbf{x}) - a_2(\mathbf{x}) - a_3(\mathbf{x}) \geq 0. & (\text{with } \mathcal{S}_1 = \{1\}, \mathcal{S}_2 = \{2,3\}). \end{cases} \tag{3.6}$$

Each big left brace indicates a system of inequalities all of which should hold, thus essentially the conjunction of the individual inequalities contained within it. We can interpret of the subsets of $\mathcal{P}$ as certificates. In order for the output of Eq. (3.4) to be positive, we need at least one certificate $\mathcal{S}_1$ (one subset of $\mathcal{P}$) such that for every subset $\mathcal{S}_2$ of $\mathcal{N}$, $w_0 + \sum_{k\in\mathcal{S}_1} a_k(\mathbf{x}) - \sum_{k\in\mathcal{S}_2} a_k(\mathbf{x}) \geq 0$. The two sets of inequalities show the choices of subsets of $\mathcal{N}$ for each of the two possible choices of $\mathcal{S}_1$ (i.e., either $\emptyset$ or $\{1\}$). The above conditions represent a disjunction of conjunctions.

Similarly, employing the first and third conditions of the theorem to our running example gives us:

$$\begin{cases} w_0 \geq 0, & \text{or} \quad w_0 + a_1(\mathbf{x}) \geq 0 \\ w_0 - a_2(\mathbf{x}) \geq 0, & \text{or} \quad w_0 + a_1(\mathbf{x}) - a_2(\mathbf{x}) \geq 0 \\ w_0 - a_3(\mathbf{x}) \geq 0, & \text{or} \quad w_0 + a_1(\mathbf{x}) - a_3(\mathbf{x}) \geq 0 \\ w_0 - a_2(\mathbf{x}) - a_3(\mathbf{x}) \geq 0, & \text{or} \quad w_0 + a_1(\mathbf{x}) - a_2(\mathbf{x}) - a_3(\mathbf{x}) \geq 0. \end{cases} \tag{3.7}$$

Note that unlike the previous case, this gives us a condition that is a conjunction of disjunctions.

3.1.2.1 Proof of the expressiveness theorem

In this section, we prove our main theorem, Theorem 3.1. Let us prove that condition 1 implies condition 2 first. We can construct a subset $\mathcal{S}_1^*$ of $\mathcal{P}$

$$\mathcal{S}_1^* = \{k : k \in \mathcal{P} \text{ and } a_k(\mathbf{x}) \geq 0\},$$

such that

$$\sum_{k \in \mathcal{P}} \text{rect}(a_k(\mathbf{x})) = \sum_{k \in \mathcal{S}_1^*} a_k(\mathbf{x}).$$

The example $\mathbf{x}$ is classified as positive implies that

$$w_0 + \sum_{k \in \mathcal{S}_1^*} a_k(\mathbf{x}) \geq \sum_{k \in \mathcal{N}} \text{rect}(a_k(\mathbf{x})).$$

For any subset $\mathcal{S}_2$ of $\mathcal{N}$, we have

$$\sum_{k \in \mathcal{N}} \text{rect}(a_k(\mathbf{x})) \geq \sum_{k \in \mathcal{S}_2} \text{rect}(a_k(\mathbf{x})) \geq \sum_{k \in \mathcal{S}_2} a_k(\mathbf{x}).$$

Therefore, for any subset $\mathcal{S}_2$ of $\mathcal{N}$,

$$w_0 + \sum_{k \in \mathcal{S}_1^*} a_k(\mathbf{x}) \geq \sum_{k \in \mathcal{S}_2} a_k(\mathbf{x}).$$

Now, we need to show that condition 2 implies condition 1. Assume there is a subset $\mathcal{S}_1$ of $\mathcal{P}$ such that for any subset $\mathcal{S}_2$ of $\mathcal{N}$, $w_0 + \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x}) \geq \sum_{k \in \mathcal{S}_2} a_k(\mathbf{x})$. Let us define a specific subset $\mathcal{S}_2^*$ of $\mathcal{N}$,

$$\mathcal{S}_2^* = \{k : k \in \mathcal{N} \text{ and } a_k(\mathbf{x}) \geq 0\},$$

such that

$$\sum_{k \in \mathcal{N}} \text{rect}(a_k(\mathbf{x})) = \sum_{k \in \mathcal{S}_2^*} a_k(\mathbf{x}).$$

We know that

$$\sum_{k \in \mathcal{P}} \text{rect}(a_k(\mathbf{x})) \geq \sum_{k \in \mathcal{S}_1} \text{rect}(a_k(\mathbf{x})) \geq \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x})$$

and

$$w_0 + \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x}) \geq \sum_{k \in \mathcal{S}_2^*} a_k(\mathbf{x}).$$

Therefore,

$$w_0 + \sum_{k \in \mathcal{P}} \text{rect}(a_k(\mathbf{x})) \geq \sum_{k \in \mathcal{N}} \text{rect}(a_k(\mathbf{x})),$$

which means that the decision function y in Eq. (3.3) is positive.

That condition 1 implies condition 3 holds by virtue of the first part of the previous step. We only need to prove that condition 3 implies 1 here. Assume for all subset $\mathcal{S}_2$ of $\mathcal{N}$, there is a subset $\mathcal{S}_1$ of $\mathcal{P}$ such that $w_0 + \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x}) - \sum_{k \in \mathcal{S}_2} a_k(\mathbf{x}) \geq 0$. Use the same $\mathcal{S}_2^*$ defined in previous step

$$\begin{aligned} w_0 + \sum_{k \in \mathcal{P}} \text{rect}(a_k(\mathbf{x})) &\geq w_0 + \sum_{k \in \mathcal{S}_1} \text{rect}(a_k(\mathbf{x})) \\ &\geq w_0 + \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x}) \\ &\geq \sum_{k \in \mathcal{S}_2^*} a_k(\mathbf{x}) \\ &= \sum_{k \in \mathcal{N}} \text{rect}(a_k(\mathbf{x})). \end{aligned}$$

Therefore, the decision function y in Eq. (3.3) is positive. □

3.1.2.2 Discussion

The only difference between the second and the third conditions of the theorem is the order of the universal and existential quantifiers over the positive and negative hidden units, $\mathcal{P}$ and $\mathcal{N}$, respectively. More importantly, in both cases, the inequality condition over the subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ is identical. Normally, swapping the order of the quantifiers does not give us an equivalent statement, but here, we see that doing so retains meaning because, in both cases, the output is positive for the corresponding input.

For any subsets $\mathcal{S}_1 \subseteq \mathcal{P}$ and $\mathcal{S}_2 \subseteq \mathcal{N}$, we can write the inequality condition as a Boolean function $B_{\mathcal{S}_1,\mathcal{S}_2}$:

$$B_{\mathcal{S}_1,\mathcal{S}_2}(\mathbf{x}) = \begin{cases} \text{true}, & w_0 + \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x}) - \sum_{k \in \mathcal{S}_2} a_k(\mathbf{x}) \geq 0 \\ \text{false}, & w_0 + \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x}) - \sum_{k \in \mathcal{S}_2} a_k(\mathbf{x}) < 0 \end{cases}. \tag{3.8}$$

If the sizes of the positive and negative subsets are n_1 and n_2, respectively (i.e, $n_1 = |\mathcal{P}|$ and $n_2 = |\mathcal{N}|$), then we know that $\mathcal{P}$ has 2^{n_1} subsets and $\mathcal{N}$ has 2^{n_2} subsets. Thus, there are $2^{n_1+n_2}$ such Boolean functions. Then, by virtue of conditions 1 and 2 of Theorem 3.1, we have[1]

[1]We write y as a Boolean with $y = 1$ and $y = -1$ representing true and false, respectively.

$$y = \vee_{\mathcal{S}_1 \subseteq \mathcal{P}} \left[\wedge_{\mathcal{S}_2 \subseteq \mathcal{N}} B_{\mathcal{S}_1, \mathcal{S}_2}(\mathbf{x}) \right],$$

where $\wedge_{\mathcal{S}_2}$ indicates a conjunction over all different subsets $\mathcal{S}_2$ of $\mathcal{N}$, and $\vee_{\mathcal{S}_1}$ indicates a disjunction over all different subsets $\mathcal{S}_1$ of $\mathcal{P}$. This expression is in the disjunctive normal form (DNF), where each conjunct contains 2^{n_2} B's, and there are 2^{n_1} such terms. Since each B simplifies into a hyperplane in the input space, this characterizes the decision boundary of the ReLU network as a DNF expression over these hyperplane decisions.

Similarly, by conditions 1 and 3, we have $y = \wedge_{\mathcal{S}_2} \left[\vee_{\mathcal{S}_1} B_{\mathcal{S}_1, \mathcal{S}_2}(\mathbf{x}) \right]$. This is in the conjunctive normal form (CNF), where each disjunctive clause contains 2^{n_1} Boolean values, and there are 2^{n_2} such clauses.

A corollary is that if the hidden units of the ReLU network are all positive (or negative), then the equivalent threshold network is a pure disjunction (or conjunction).

3.2 Comparing ReLU and threshold networks

In the previous section, we saw that ReLU networks could express Boolean functions that correspond to much larger threshold networks. Of course, threshold activations are not generally used in applications; nonetheless, they are well understood theoretically, *and* they emerge naturally as a result of the analysis above. Using threshold functions as a vehicle to represent the decision boundaries of ReLU networks, naturally leads to two related questions that we will address in this section. First, given an arbitrary ReLU network, can we construct an equivalent threshold network? Second, given an arbitrary threshold network, how can we represent it using ReLU network?

3.2.1 Converting from ReLU to threshold

Theorem 3.1 essentially gives us a constructive way to represent an arbitrary two-layer ReLU network given in Eq. (3.3) as a three-layer threshold network. For every choice of the subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ of the positive and negative units, we can define a Boolean function $B_{\mathcal{S}_1, \mathcal{S}_2}$ as per Eq. (3.8). By definition, each of these is a threshold unit, giving us $2^{n_1+n_2}$ threshold units in all. (Recall that n_1 and n_2 are the sizes of $\mathcal{P}$ and $\mathcal{N}$, respectively.) Since the decision function is a CNF or a DNF over these functions, it can be represented using a two-layer network over the B's, giving us three layers in all. We put all $2^{n_1+n_2}$ threshold units in the first hidden layer, separated into 2^{n_1} groups, with each group comprising of

2^{n_2} units.

Figure 3.2 shows the threshold network corresponding to our running example from Eq. (3.4). For brevity, we use the notation $B_{i,j}$ to represent the first hidden layer, with i and j indexing over the subsets of $\mathcal{P}$ and $\mathcal{N}$, respectively. The B's can be grouped into two groups, with units in each group sharing the same subset $\mathcal{S}_1$ but with different $\mathcal{S}_2$. Note that, these nodes are linear threshold units corresponding to the inequalities in Eq (3.5) and (3.6). In the second hidden layer, we have one threshold unit connected to each group of units in the layer below. The weight for each connection unit is 1, and the bias is $2^{n_2} - 1$, effectively giving us the conjunction of the previous layer nodes. The second hidden layer has 2^{n_1} such units. Finally, we have one unit in the output layer, with all weights being 1 and bias being $1 - 2^{n_1}$, simulating a disjunction of the decisions of the layer below. As discussed in the previous section, we can also construct a threshold network using CNFs, with $2^{n_1+n_2}$ units in the first hidden layer, and 2^{n_2} units in the second hidden layer.

3.2.2 Boolean expressiveness of ReLU networks

Theorem 3.1 shows that for an arbitrary two-layer ReLU network, we *can always* represent it using a three-layer threshold network in which the number of threshold units is exponentially more than the number of ReLUs. However, this does not imply that we need that many thresholds units. A natural question to ask is: can we represent the same ReLU

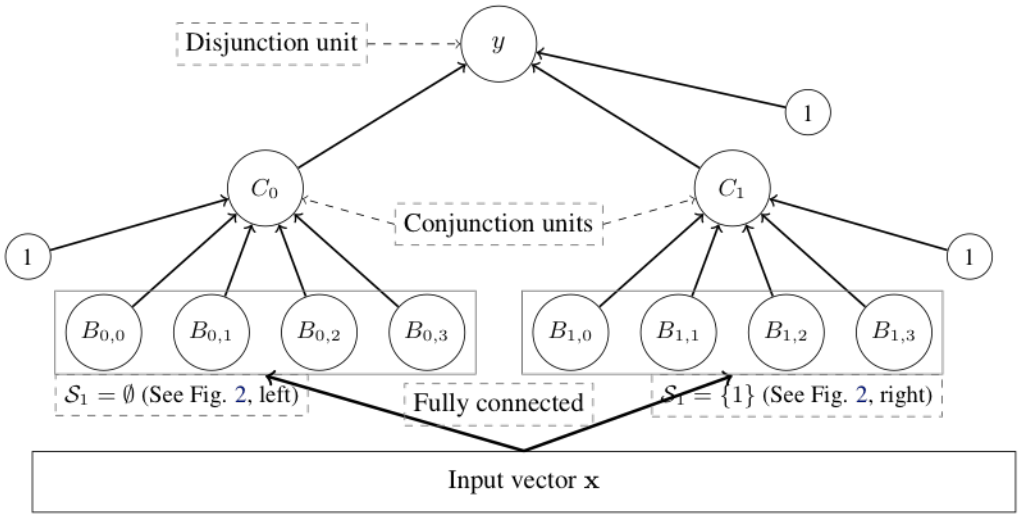

Figure 3.2. A threshold network corresponding to the running example. The dotted boxes label the various components of the network. See the text above for details.

network without the expense of exponential number of threshold units? In general, the answer is no, because there are families of ReLU network that need at least an exponential number of threshold units to represent.

We can formalize this for the case of ReLU networks where the hidden nodes are either all positive or negative – that is, either $\mathcal{P}$ or $\mathcal{N}$ is the empty set. We restrict ourselves to this set because we can represent such networks using a two-layer threshold network rather than the three-layer one in the previous section. We will consider the set of all Boolean functions in d dimensions expressed by such a ReLU network with n hidden units. Let $\Gamma_R(n, d)$ represent this set. Similarly, let $\Gamma_T(n, d)$ denote the set of such functions expressed by threshold networks with n hidden units. We can summarize the Boolean expressiveness of ReLU networks via the following two-part theorem:

Theorem 3.2. *For rectifier networks with $n > 1$ hidden units, all of which are either positive or negative, and for all input dimensionalities $d \geq n$, we have*

1. $\Gamma_R(n, d) \subseteq \Gamma_T(2^n - 1, d)$, *and,*

2. $\Gamma_R(n, d) \not\subset \Gamma_T(2^n - 2, d)$.

Proof. The first part of this theorem says that every rectifier network with n hidden units that are all positive or negative can be represented by a threshold network with $2^n - 1$ hidden units. This is a direct consequence of the main theorem.

The second part of the theorem says that for any n, there are families of rectifier networks whose equivalent threshold network will need an exponential number of hidden threshold units. We prove this assertion constructively by providing one such rectifier network. Consider the decision function of a two-layer rectifier network

$$y = \text{sgn}[-1 + \text{rect}(x_1) + \text{rect}(x_2) + \cdots + \text{rect}(x_n)],$$

where x_i is the i^{th} component of the input vector (recall that the dimensionality of the input $d \geq n$). From Theorem 3.1, the decision boundary of this network can be determined by $2^n - 1$ hyperplanes of the form $-1 + \sum_{i \in \mathcal{S}} x_i = 0$, each of which corresponds a *nonempty* subset $\mathcal{S} \in [n]$. The output y is positive if any of these hyperplanes classify the example as positive.

To prove that we need at least $2^n - 1$ threshold units to represent the same decision boundary, it suffices to prove that each of the $2^n - 1$ hyperplanes are needed to define the decision boundary.

Consider any hyperplane defined by *nonempty* subset $\mathcal{S} \in [n]$ whose cardinality $s = |\mathcal{S}|$. Let $\bar{\mathcal{S}}$ be the complement of $\mathcal{S}$. Consider an input vector $\mathbf{x}$ that satisfies the following condition if $s > 1$:

$$\begin{cases} 1/s < x_i < 1/(s-1), & \text{if } i \in \mathcal{S} \\ x_i < -1/(s-1), & \text{if } i \in \bar{\mathcal{S}}. \end{cases}$$

For those subsets $\mathcal{S} = \{x_j\}$ whose cardinality is one, we can let $x_j > 1$ and all the other components $x_i < -x_j$. Clearly, the rectifier network will classify the example as positive. Furthermore, by construction, it is clear that $-1 + \sum_{i \in \mathcal{S}} x_i > 0$, and for all other subset $\mathcal{S}' \in [n]$, $-1 + \sum_{i \in \mathcal{S}'} x_i < 0$. In other words, *only* the selected hyperplane will classify the input as a positive one. That is, this hyperplane is required to define the decision boundary because without it, the examples in the above construction will be incorrectly classified by the threshold network.

Therefore we have identified the decision boundary of the given rectifier network as a polytope with *exactly* $2^n - 1$ faces, by constructing $2^n - 1$ hyperplanes using $2^n - 1$ *nonempty* subsets $\mathcal{S} \in [n]$. To complete the proof, we need to show that other construction methods cannot do better than our construction, i.e., achieving the same decision boundary with less hidden threshold units. To see this, note that for each face of the decision polytope, one needs a hidden threshold unit to represent it. Therefore no matter how we construct the threshold network, we need at least $2^n - 1$ hidden threshold units. □

Let us look at an intuitive explanation of this result. This theorem tells us that the upper bound on the number of threshold units corresponding to the ReLU network is a tight one. In other words, not only can ReLU networks express Boolean functions that correspond to much larger threshold networks, there are some ReLU networks that can *only* be expressed by such large networks! This theorem may give us some intuition into the successes of ReLU networks.

Note, however, that this theorem does not resolve the question of what fraction of all ReLU networks can be expressed using fewer than exponential number of hidden threshold units. Indeed, if the inputs were only Boolean and the output was allowed to be real-valued, then [60, Theorem 6] show that a two-layer ReLU network can be simu-

lated using only a quadratic number of threshold units. (In contrast, Theorem 3.2 above considers the case of real-valued inputs, but Boolean outputs.)

3.2.3 Converting thresholds to ReLUs

So far, we have looked at threshold networks corresponding to ReLU networks. The next question we want to answer is under what condition we can use ReLUs to represent the same decision boundary as a threshold network. In this section, we show a series of results that address various facets of this question. The longer proofs are in the appendices.

First, with no restrictions in the number of ReLUs in the hidden layer, we can always construct a rectifier network that is equivalent to a threshold network. In fact, we have the following lemma:

Lemma 3.1. *Any threshold network with n units can be approximated to arbitrary accuracy by a rectifier network with $2n$ units.*

Proof. Consider a threshold unit with weight vector $\mathbf{v}$ and bias d, we have

$$\text{sgn}(\mathbf{v} \cdot \mathbf{x} + d) \simeq \frac{1}{\epsilon}\big[\text{rect}(\mathbf{v} \cdot \mathbf{x} + d + \epsilon) - \text{rect}(\mathbf{v} \cdot \mathbf{x} + d - \epsilon)\big] - 1,$$

where ϵ is an arbitrary small number that determines the approximation accuracy. □

This result is akin to the previous simulation results [57] that compares threshold and sigmoid networks.

Given the exponential increase in the size of the threshold network to represent a ReLU network (Theorem 3.2), a natural question is whether we can use only logarithmic number of ReLUs to represent any arbitrary threshold network. In the general case, the following lemma points out that this is not possible.

Lemma 3.2. *There exists a two-layer network with n hidden threshold units for which it is not possible to construct an equivalent two-layer ReLU network with less hidden units.*

Proof. In this section, we provide a simple example of a two-layer threshold network, whose decision boundary cannot be represented by a two-layer rectifier network with fewer hidden units. Consider a threshold network

$$y = \text{sgn}[n - 1 + \text{sgn}(x_1) + \text{sgn}(x_2) + \cdots + \text{sgn}(x_n)],$$

where x_i is the i^{th} component of the input vector (here we assume the dimensionality of the input $d \geq n > 1$). It is easy to see that the decision function of this network is positive, if, and only if at least one of the component x_i is nonnegative. From a geometric point of view, each x_i defines a hyperplane $x_i = 0$. Let H_1 be the set of n such hyperplanes,

$$H_1 = \{x_i = 0 : i \in [n]\}$$

These n hyperplanes form 2^n hyperoctants, and only one hyperoctant gets negative label.

Now, suppose we construct a two-layer network with m rectifier units that has the same decision boundary as the above threshold network, and it has the form

$$y = \text{sgn}\left(w_0 + \sum_{k \in \mathcal{P}} \text{rect}(a_k(\mathbf{x})) - \sum_{k \in \mathcal{N}} \text{rect}(a_k(\mathbf{x}))\right),$$

using the same notations as in Theorem 3.1. From the theorem, we know the decision boundary of the above equation is determined by 2^m hyperplane equations, and these 2^m hyperplanes form a set H_2,

$$H_2 = \left\{w_0 + \sum_{k \in \mathcal{S}_1} a_k(\mathbf{x}) - \sum_{k \in \mathcal{S}_2} a_k(\mathbf{x}) = 0 : \mathcal{S}_1 \subseteq \mathcal{P}, \mathcal{S}_2 \subseteq \mathcal{N}\right\}.$$

Because we assume the rectifier network has the same decision function as the threshold network, we have

$$H_1 \subseteq H_2.$$

Note that the hyperplanes in H_1 have normal vectors independent of each other, which means there are at least n hyperplanes in H_2 such that their normal vectors are independent. Recall that $a_k(\mathbf{x}) = \mathbf{u}_k \cdot \mathbf{x} + b_k$, so all normal vectors for hyperplanes in H_2 can be expressed using linear combinations of m vectors $\mathbf{u}_1, \ldots, \mathbf{u}_m$. Since these m vectors together define the n orthogonal hyperplanes in H_1, it is impossible to have $m < n$. In other words, for this threshold network, the number of hidden units cannot be reduced by any conversion to a ReLU network. □

This lemma, in conjunction with Theorem 3.2, effectively points out that by employing a rectifier network, we are exploring a *subset* of much larger threshold networks. Furthermore, despite the negative result of the lemma, in the general case, we can identify certain specific threshold networks that can be compressed into logarithmically smaller ones using ReLUs. Suppose we wish to compress a two-layer threshold network with

three sign hidden units into a ReLU network with $\lceil \log_2 3 \rceil = 2$ hidden units. The sign network can be represented by

$$y = \text{sgn}(2 + \text{sgn}\,(\mathbf{v}_1 \cdot \mathbf{x} + d_1) + \text{sgn}\,(\mathbf{v}_2 \cdot \mathbf{x} + d_2) + \text{sgn}\,(\mathbf{v}_3 \cdot \mathbf{x} + d_3)).$$

Suppose one of the weight vectors can be written as the linear combination of the other two but its bias can not. That is, for some p and q, if $\mathbf{v}_3 = p\mathbf{v}_1 + q\mathbf{v}_2$ and $d_3 \neq pd_1 + qd_2$. Then, we can construct the following equivalent ReLU network that is equivalent:

$$y = \text{sgn}\,(-1 + \text{rect}(\mathbf{u}_1 \cdot \mathbf{x} + b_1) + \text{rect}(\mathbf{u}_2 \cdot \mathbf{x} + b_2))\,,$$

where

$$\begin{aligned}
\mathbf{u}_1 &= pr\mathbf{v}_1, \\
\mathbf{u}_2 &= qr\mathbf{v}_2, \\
b_1 &= prd_1 + 1, \\
b_2 &= qrd_2 + 1, \\
r &= \frac{1}{d_3 - pd_1 - qd_2}.
\end{aligned}$$

This equivalence can be proved by applying Theorem 3.1 to the constructed ReLU network. It shows that in two dimensions, we can use two ReLUs to represent three linearly independent sign units.

We can generalize this result to the case of a two-layer threshold network with 2^n hidden threshold units that represents a disjunction over the hidden units. The goal is to find that under what condition we can use only n rectifier units to represent the same decision. To do so, we will use binary encoding matrix T_n of size $n \times 2^n$ whose i^{th} column is the binary representation of $i - 1$. For example, the binary encoding matrix for $n = 3$ is given by T_3,

$$T_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Lemma 3.3. *Consider a two-layer threshold network with 2^n threshold units in the hidden layer whose output represents a disjunction over the hidden units, i.e., the final output is positive if and only if at least one of the hidden-unit outputs is positive. That is,*

$$y = \text{sgn}\left(2^n - 1 + \sum_{k=1}^{2^n} \text{sgn}\,(\mathbf{v}_k \cdot \mathbf{x} + d_k)\right). \tag{3.9}$$

This decision can be represented using a two-layer rectifier network with n hidden units, if the weight parameters of the threshold units can be factored in the following form:

$$\begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_{2^n} \\ d_1 & \cdots & d_{2^n} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_n & \mathbf{0} \\ b_1 & \cdots & b_n & w_0 \end{bmatrix} \begin{bmatrix} T_n \\ \mathbf{e}_{2^n} \end{bmatrix}, \tag{3.10}$$

where $\mathbf{e}_{2^n}$ *is a* 2^n *dimensional row vector of all ones and* $\mathbf{0}$ *is a vector of all zeros.*

Proof. If the weight parameters $\mathbf{v}_k$ and d_k can be written in the form as in Eq. (3.10), then we can construct the two-layer rectifier network,

$$y = \operatorname{sgn}\big[w_0 + \sum_{k=1}^{n} \operatorname{rect}(\mathbf{u}_k \cdot \mathbf{x} + b_k)\big]. \tag{3.11}$$

Then by virtue of Theorem 3.1, the decision boundary of the rectifier network in Eq. (3.11) is the same as the decision boundary of the threshold network in Eq. (3.9). □

Note that this lemma only identifies sufficient conditions for the logarithmic reduction in network size. Identifying both necessary and sufficient conditions for such a reduction is an open question.

3.3 Hidden layer equivalence

Lemma 3.3 studies a specific threshold network, where the output layer is a disjunction over the hidden layer units. For this network, we can define a different notion of equivalence between networks by studying the hidden layer activations. We do so by interpreting the hidden layer state of the network as a specific kind of a multiclass classifier that either rejects inputs or labels them. If the output is negative, then clearly none of the hidden layer units are active, and the input is rejected. If the output layer is positive, then at least one of the hidden layer units is active, and the multiclass label is given by the maximum scoring hidden unit, namely $\arg\max_k \mathbf{v}_k \cdot \mathbf{x} + d_k$.

For threshold networks, the number of hidden units is equal to the number of classes. The goal is to learn the same concept with rectifier units, hopefully with fewer rectifier units than the number of classes. Suppose a ReLU network has n hidden units, then its hidden layer prediction is the highest scoring hidden unit of the corresponding threshold network that has 2^n hidden units. We now define *hidden layer equivalence* of two networks as follows: A threshold network and a ReLU network are equivalent if both their hidden layer predictions are identical.

We already know from Lemma 3.3 that if the weight parameters of the true concept satisfy Eq. (3.10), then instead of learning 2^n threshold units we can just learn n rectifier units. For simplicity, we write Eq. (3.10) as $V = UT$ where

$$V = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_{2^n} \\ d_1 & \cdots & d_{2^n} \end{bmatrix} \quad U = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_n & \mathbf{0} \\ b_1 & \cdots & b_n & w_0 \end{bmatrix}$$

and

$$T = \begin{bmatrix} T_n \\ \mathbf{e}_{2^n} \end{bmatrix}.$$

For simplicity of notation, we will assume that the input features $\mathbf{x}$ includes a constant bias feature in the last position. Thus, the vector $V^T\mathbf{x}$ represents the preactivation score for each class.

Now, we consider threshold networks with parameters such that there is *no* ReLU (defined by the matrix U) that satisfies this condition. Instead, we find a rectifier network with parameters U that satisfies the following condition:

$$U = \text{argmin}_U \|(V - UT)^T\|_\infty, \tag{3.12}$$

Here $\|\cdot\|_\infty$ is the induced infinity norm, defined for any matrix A as $\|A\|_\infty = \sup_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty}$.

If we have a matrix U such that V and UT are close in the sense of induced infinity norm, then we have the following about their equivalence.

Theorem 3.3. *If the true concept of a 2^n-class classifier is given by a two-level threshold network in Eq.* (3.9)*, then we can learn a two-layer rectifier network with only n hidden units of the form in Eq.* (3.11) *that is hidden layer equivalent to it, if for any example* $\mathbf{x}$*, we have*

$$\|(V - UT)^T\|_\infty \leq \frac{\gamma(\mathbf{x})}{2\|\mathbf{x}\|_\infty}, \tag{3.13}$$

where $\gamma(\mathbf{x})$ is the multiclass margin for $\mathbf{x}$*, defined as the difference between its highest score and second-highest scoring classes.*

Proof. The proof is based on the intuition that for hidden layer equivalence, as defined above, *only* requires that the highest scoring label needs to be the same in the two networks rather than the actual values of the scores. If V and UT are closed in the sense of induced infinity norm, then the highest scoring hidden unit will be invariant regardless of which network is used.

More specifically, let us define $\epsilon = \|(V - UT)^T\|_\infty \leq \frac{\gamma(x)}{2\|x\|_\infty}$. From the definition of the L_∞ vector norm, we have

$$\|(V - UT)^T x\|_\infty \geq |((V - UT)^T x)_k|$$

for all x and all k. The subscript k labels the k^{th} component of the vector. From the definition of the induced norm we have

$$\|(V - UT)^T x\|_\infty \leq \epsilon \|x\|.$$

Combining the above two inequalities we have

$$|((UT)^T x)_k - (V^T x)_k| \leq \epsilon \|x\|$$

for all x and all k. Assuming k^* is the highest scoring unit, for k^* we have

$$(V^T x)_{k^*} - ((UT)^T x)_{k^*} \leq \epsilon \|x\|.$$

For any other $k' \neq k^*$, we have

$$((UT)^T x)_{k'} - (V^T x)_{k'} \leq \epsilon \|x\|.$$

From the definition of the margin $\gamma(x)$, we also know that

$$(V^T x)_{k^*} - (V^T x)_{k'} \geq \gamma(x) \geq 2\epsilon \|x\|.$$

Combining the above three inequalities, we have

$$((UT)^T x)_{k'} \leq ((UT)^T x)_{k^*},$$

which means if k^* is the correct class with the highest score according to the weight parameters V, it will still be the highest scoring class according to the weight parameters UT, even if $V \neq UT$. □

3.4 Experiments

We have seen that every two-layer rectifier network expresses the decision boundary of a three-layer threshold network. If the output weights of the former are all positive, then a two-layer threshold network is sufficient. (See the discussion in Section 3.2.2.)

However, the fact that rectifier network can express the same decision boundary more compactly does not guarantee learnability because of optimization issues. Specifically, in this section, we study the following question using synthetic data: *Given a rectifier network and a threshold network with same decision boundary, can we learn one using the data generated from another using backpropagation?*

3.4.1 Data generation

We use randomly constructed two-layer rectifier networks to generate labeled examples. To do so, we specify various values of the input dimensionality and the number of hidden ReLU units in the network. Once we have the network, we randomly generate the input points and label them using the network. Using generated data, we try to recover both the rectifier network and the threshold network, with different number of hidden units. We considered input dimensionalities 3, 10, and 50, and in each case, using 3 or 10 hidden units. This gave us six networks in all. For each network, we generated 10000 examples and 1500 of which are used as test examples.

3.4.2 Results and analysis

For each dataset, we compare three different network architectures. The key parameter that varies across datasets is n, the number of hidden ReLU units in the network that generated the data. The first setting learns using a ReLU network with n hidden units. The second setting uses the activation function $\tanh(cx)$, which we call the compressed tanh activation. For large values of c, this effectively simulates the threshold function. In the second setting, the number of hidden units is still n. The final setting learns using the compressed tanh, but with 2^n hidden units following Section 3.1.2.

Figure 3.3 shows the results of the six datasets. These results verify several aspects of our theory. First, learning using ReLUs always succeeds with low error (purple bars, left). This is expected – we know that our hypothesis class can express the true concept and training using backpropagation can successfully find it. Second, learning using compressed tanh with the same number of units cannot recover the true concept (red bars, middle). This performance drop is as expected since compressed tanh is just like the sign activation, and we know in this case we need exponentially more hidden units.

Finally, the performances of learning using the exponential number of compressed tanh

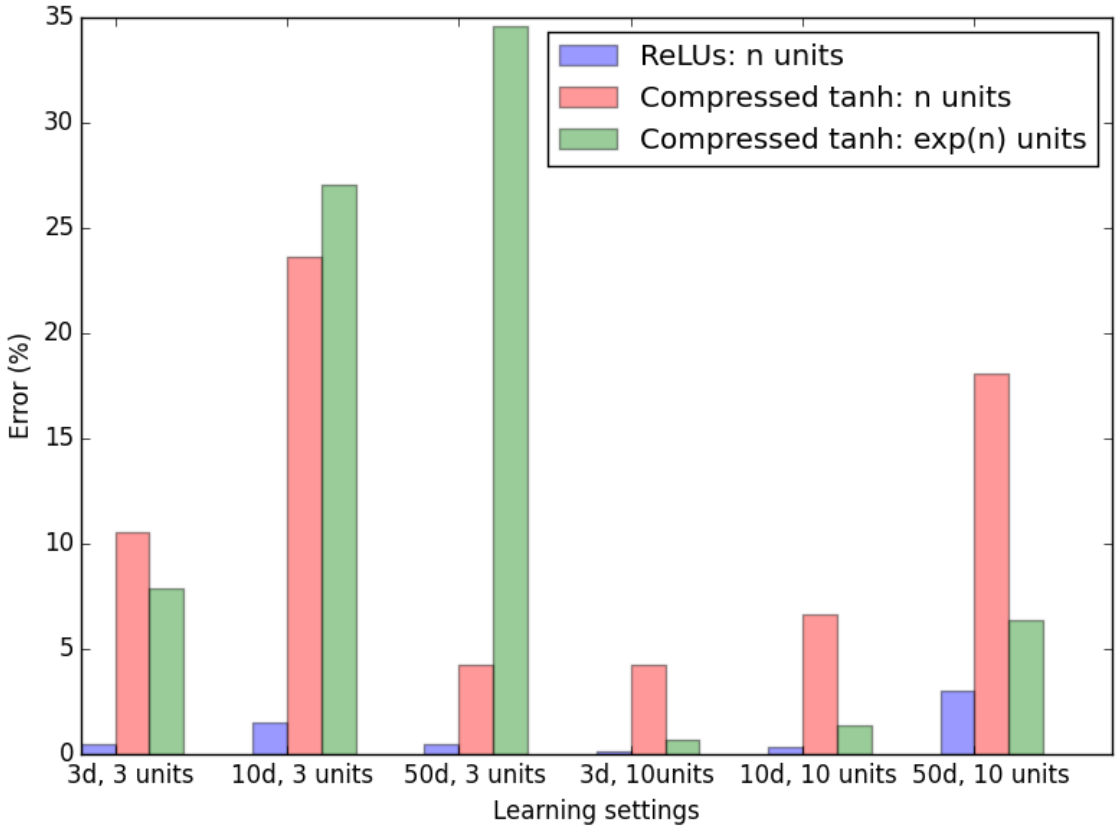

Figure 3.3. Test errors for different learning settings. The x-axis specifies the number of ReLUs used for data generalization and the dimensionality. Each dataset is learned using ReLU and compressed tanh activations with different number of hidden units. Learning rate is selected with cross-validation from $\{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. L_2-regularization coefficient is 10^{-4}. We use early-stopping optimization with a maximum of 1000 epochs. The minibatch size is 20. For the compressed tanh, we set $c = 10000$.

(green bars, right) are not always good.[2] In this case, from the analysis in Section 3.1, we know the hypothesis can certainly express the true concept; yet learning does not always succeed! In fact, for the first three groups, where we have three ReLUs for data generation, the error for the learned classifier is rather large, suggesting that even though the true concept can be expressed, it is not found by backpropagation. For the last three groups, where we have 10 hidden ReLUs for data generation, using the exponential number of compressed tanh does achieve better performance. We posit that this incongruence is due to the interplay between the nonconvexity of the objective function and the fact that the set of functions expressed by threshold functions is larger (a consequence of Lemma 3.2).

3.5 Conclusions

In this chapter, we have presented a novel analysis of the expressiveness of rectifier neural networks. Specifically, for binary classification we showed that even though the decision boundary of two-layer rectifier network can be represented using threshold unit network, the number of threshold units required is exponential. Further, while a corresponding general logarithmic reduction of threshold units is not possible, for specific networks, we characterized sufficient conditions for reducing a threshold network to a much smaller rectifier network. We also presented a relaxed condition where we can approximately recover a rectifier network that is hidden layer equivalent to an exponentially larger threshold network.

Our work presents a natural next step: can we use the equivalence of the expressiveness results given in this chapter to help us study the sample complexity of rectifier networks? Another open question is the generalization of these results to deep networks, which is covered in Chapter 6. Finally, from our experiments, we see that expressiveness is not enough to guarantee learnability. Studying the interplay of expressiveness, sample complexity, and the convexity properties of the training objective function for rectifier networks represents an exciting direction of future research.

---

[2]We also evaluated the performance on the training set. The training errors for all six datasets for all three learning scenarios are very close to test error, suggesting that over-fitting is not an issue.

CHAPTER 4

LEARNING LINEAR CONSTRAINTS FOR STRUCTURED PREDICTION USING RECTIFIER NETWORKS

Various NLP tasks can be formulated as structured prediction problems where outputs are joint assignments to interdependent variables. Past work has shown that domain knowledge, framed as constraints over the outputs, can help improve predictive accuracy. However, designing proper constraints often relies on domain expertise. In this chapter, we study the problem of learning such constraints. We use the theoretical results from Chapter 3 to learn constraints from training data of a structured prediction problem. We frame it as training a two-layer rectifier network to identify valid structures or substructures and show a construction for converting a trained network into a system of linear constraints over the inference variables. Our experiments on several NLP tasks show that the learned constraints can effectively improve prediction quality.

4.1 Constraints in structured prediction

As we discussed in Section 2.2, the interactions among output variables in a structured prediction problem can be modeled either as feature factors or as constraints. In principle every hard constraint can be equivalently modeled as a feature factor: the score of the constraint-violating assignments of the variables is negative infinity. In practice it is often beneficial to model some interactions as constraints rather than as feature factors. This is because feature design is limited by computational capabilities. Large feature factors and global features, which involve multiple inference variables, often make the inference and training process computationally intractable. On the other hand, it is often possible to enforce constraints over large number of variables during inference, while still admit efficient training process.

Recent progress on neural networks for NLP has partially solved the feature engineer-

ing problem with learned representations of the underlying text [31, 32]. However, the design of constraints largely remains task-specific and often requires manual effort. In this paper we ask the question: *can we use neural network methods to discover constraints from data automatically, and use them to predict structured outputs?* We provide a general framework for discovering constraints as linear inequalities over the output variables of a structured prediction problem. These constraints can improve the performance of an already trained model, or be integrated into the learning process for global training.

A system of linear inequality constraints represents a convex polytope (bounded or unbounded) and can be naturally expressed as a two-layer threshold network, i.e., a network with one hidden layer of linear threshold units and an output layer with a single threshold unit. This two-layer threshold network will predict 1 or -1 depending on whether the system of linear inequalities is satisfied or not. In principle, we could try to train such a threshold network representing a set of linear inequalities. However, the zero-gradient nature of the threshold activation function prohibits using backpropagation for training.

In this chapter, we train a specific binary classifier that is a two-layer rectifier network to identify valid structures or substructures. This network also contains a single linear threshold output unit, but in the hidden layer, it contains rectified linear units (ReLUs). Chapter 3 shows that a two-layer rectifier network constructed in such a way is equivalent to a threshold network, thus representing the same set of linear inequalities as the threshold network with far fewer hidden units than the equivalent threshold network.

As we saw in Section 2.2, linear constraints thus obtained can be integrated with an existing model in multiple ways, depending on the inference algorithm used by the original model. Therefore the framework proposed in this chapter can be viewed as a general approach to improve structured prediction using linear inequality constraints.

We perform one synthetic and two information extraction experiments to verify our proposed idea. In the synthetic experiment, we randomly generate many ILP instances with shared (hidden) constraints. The goal is to recover the constraints from the solution of the examples. The first information extraction task is to extract citation fields like authors, journals, and dates from a bibliography entry. We treat it as a sequence labeling problem and show that learned constraints can improve an existing first-order Markov model. The second one is an entity and relation extraction task, in which we try to label the entity

candidates and identify relations between them. Unlike the citation field extraction task, the output is not a labeled sequence but a labeled directed graph. In this task, we show that the learned constraints can be used while training the model to improve prediction.

In summary, the contributions of this chapter are the following:

1. We propose a method for learning constraints for structured prediction using rectifier neural networks. Our method is driven by a novel construction for converting the learned rectifier networks into linear inequalities.

2. We show via experiments that adding learned constraints during prediction time can substantially improve the performance of trained models. The learned constraints can also be used to train a better model.

4.2 Representing constraints

Let us revisit the formal definition of structured prediction and constraints. In a structured prediction problem, we are given an input $\mathbf{x}$ belonging to some instance space, such as sentences or documents. The goal is to predict an output $\mathbf{y} \in \mathcal{Y}_\mathbf{x}$, where $\mathcal{Y}_\mathbf{x}$ is the set of possible output structures for the input $\mathbf{x}$. The output space $\mathcal{Y}_\mathbf{x}$ has a predefined structure (e.g., trees, or labeled graphs), and the number of candidate structures in $\mathcal{Y}_\mathbf{x}$ is usually large, i.e., exponential in the size of input $\mathbf{x}$.

Inference in a structured prediction problem can be framed as an optimization problem with a linear objective function:

$$\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}_\mathbf{x}} \boldsymbol{\alpha} \cdot \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}), \tag{4.1}$$

where $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ is a feature vector representation of the input-output pair $(\mathbf{x}, \mathbf{y})$, and $\boldsymbol{\alpha}$ are learned parameters. The feature representation $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ can be designed by hand or learned using neural networks. The feasible set $\mathcal{Y}_\mathbf{x}$ is predefined and known for every $\mathbf{x}$ at both learning and inference stages. The goal of learning is to find the best parameters $\boldsymbol{\alpha}$ using training data, and the goal of inference is to solve the above argmax problem using the parameters $\boldsymbol{\alpha}$.

In this chapter, we learn constraints from training examples $\{(\mathbf{x}, \mathbf{y})\}$. Suppose we want to learn K constraints, and the k^{th} one can be represented as the Boolean function[1]:

$$c_k(\mathbf{x}, \mathbf{y}) = \begin{cases} 1, & \text{if } (\mathbf{x}, \mathbf{y}) \text{ satisfies constraint } c_k, \\ -1, & \text{otherwise.} \end{cases}$$

Then, the optimal structure $\mathbf{y}^*$ is the solution to the following optimization problem:

$$\begin{aligned} & \max_{\mathbf{y} \in \mathcal{Y}_\mathbf{x}} \boldsymbol{\alpha} \cdot \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}). \\ \text{s.t.} \quad & \forall k, c_k(\mathbf{x}, \mathbf{y}) = 1. \end{aligned}$$

We will show that such learned constraints can be beneficial for prediction performance.

4.2.1 Constraints as linear inequalities

Many constraints in NLP can be naturally expressed as linear inequalities over inference variables; several algorithms directly incorporate them into the inference process. In this chapter, we consider the constraints that can be represented as linear inequalities over some representations $\boldsymbol{\psi}(\mathbf{x}, \mathbf{y})$ of a given input-output pair. The k^{th} constraint c_k is satisfied by $(\mathbf{x}, \mathbf{y})$ if and only if the following linear inequality holds,

$$\mathbf{w}_k \cdot \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) + b_k \geq 0, \tag{4.2}$$

where the weight vector $\mathbf{w}_k$ and bias b_k are to be learned. The Boolean constraint function is therefore the linear threshold function,

$$c_k(\mathbf{x}, \mathbf{y}) = \text{sgn}\left(\mathbf{w}_k \cdot \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) + b_k\right), \tag{4.3}$$

where $\text{sgn}(\cdot)$ is the sign function: $\text{sgn}(x) = 1$ if $x \geq 0$, and -1 otherwise.

The feature representations $\boldsymbol{\psi}(\mathbf{x}, \mathbf{y})$ should not be confused with the original features $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$ used in the structured prediction model in Eq. (4.1). Hereafter, we refer $\boldsymbol{\psi}(\mathbf{x}, \mathbf{y})$ as constraint features. Constraint features should be general properties of inputs and outputs since we want to learn domain-specific constraints over them. They are design choices, and in our experiments, we will use common NLP features. In general, constraint features can also be learned using a neural network. Given a constraint feature representation $\boldsymbol{\psi}(\cdot)$, the goal is thus to learn the parameters $\mathbf{w}$'s and b's for every constraint.

[1] We use 1 to indicate true and -1 to indicate false.

4.2.2 Constraints as threshold networks

For an input $\mathbf{x}$, we say the output $\mathbf{y}$ is feasible if it satisfies constraints c_k for all $k = 1, \ldots, K$. We can define an Boolean variable $z(\mathbf{x}, \mathbf{y})$, indicating whether $\mathbf{y}$ is feasible with respect to the input $\mathbf{x}$:

$$z(\mathbf{x}, \mathbf{y}) = c_1(\mathbf{x}, \mathbf{y}) \wedge \cdots \wedge c_K(\mathbf{x}, \mathbf{y}). \tag{4.4}$$

That is, z is a conjunction of all the Boolean functions corresponding to each constraint. We can rewrite Eq. (4.4) as a linear threshold function:

$$z(\mathbf{x}, \mathbf{y}) = \operatorname{sgn}\Big(1 - K + \sum_{k=1}^{K} c_k(\mathbf{x}, \mathbf{y})\Big). \tag{4.5}$$

It is easy to see that $z(\mathbf{x}, \mathbf{y}) = 1$ if, and only if, all c_k's are 1 — precisely the definition of a conjunction. Finally, we can plug Eq. (4.3) into Eq. (4.5):

$$z = \operatorname{sgn}\Big(1 - K + \sum_{k=1}^{K} \operatorname{sgn}\left(\mathbf{w}_k \cdot \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) + b_k\right)\Big). \tag{4.6}$$

Observe that Eq. (4.6) is exactly a two-layer neural network: $\boldsymbol{\psi}(\mathbf{x}, \mathbf{y})$ is the input to the network; the hidden layer contains K linear threshold units with parameters $\mathbf{w}_k$ and b_k; and the output layer has a single linear threshold unit. This threshold neural network will predict 1 if the structure $\mathbf{y}$ is feasible with respect to input $\mathbf{x}$, and -1 if it is infeasible. In other words, constraints for structured prediction problems can be written as two-layer threshold networks. One possible way to learn constraints is thus to learn the hidden layer parameters $\mathbf{w}_k$ and b_k, with fixed output layer parameters. However, the neural network specified in Eq. (4.6) is not friendly to gradient-based learning; the $\operatorname{sgn}(\cdot)$ function has zero gradients almost everywhere. In this chapter, we explore an alternative way of learning constraints using rectifier networks rather than threshold networks.

4.2.3 Constraints as rectifier networks

We saw in the previous section that a system of linear inequalities could be represented as a two-layer threshold network. In this section, we will see a special rectifier network that is equivalent to a system of linear inequalities, and whose the parameters can be learned using backpropagation.

Consider the following two-layer rectifier network:

$$z(\mathbf{x}, \mathbf{y}) = \operatorname{sgn}\Big(1 - \sum_{k=1}^{K} \operatorname{rect}\big(\mathbf{w}_k \cdot \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) + b_k\big)\Big). \tag{4.7}$$

The input to the network is still $\boldsymbol{\psi}(\mathbf{x}, \mathbf{y})$. There are K ReLUs in the hidden layer, and one threshold unit in the output layer.

The decision boundary of this rectifier network is specified by a system of linear inequalities. In particular, we have the following theorem:

Theorem 4.1. *Consider a two-layer rectifier network with K hidden ReLUs as in Eq.* (4.7)*. Define the set* $[K] = \{1, 2, \ldots, K\}$*. The network outputs* $z(\mathbf{x}, \mathbf{y}) = 1$ *if, and only if, for every subset* $\mathcal{S}$ *of* $[K]$*, the following linear inequality holds:*

$$1 - \sum_{k \in \mathcal{S}} \big(\mathbf{w}_k \cdot \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) + b_k\big) \geq 0. \tag{4.8}$$

Proof. This theorem follows from Theorem 3.1. Here we give a direct proof. Define $a_k = \mathbf{w}_k \cdot \boldsymbol{\psi}(\mathbf{x}, \mathbf{y}) + b_k$. We first prove the "if" part of the theorem. Suppose that for any $\mathcal{S} \subseteq [K]$, $1 - \sum_{k \in \mathcal{S}} a_k \geq 0$. Thus, for a specific subset $\mathcal{S}^* = \{k \in [K] : a_k \geq 0\}$, we have $1 - \sum_{k \in \mathcal{S}^*} a_k \geq 0$. By the definition of $\mathcal{S}^*$, $\sum_{k=1}^{K} \operatorname{rect}(a_k) = \sum_{k \in \mathcal{S}*} a_k$, and therefore $1 - \sum_{k=1}^{K} \operatorname{rect}(a_k) \geq 0$. Next, we prove the "only if" part of the theorem. Suppose that $1 - \sum_{k=1}^{K} \operatorname{rect}(a_k) \geq 0$. For any $\mathcal{S} \subseteq [K]$, we have $\sum_{k=1}^{K} \operatorname{rect}(a_k) \geq \sum_{k \in \mathcal{S}} \operatorname{rect}(a_k) \geq \sum_{k \in \mathcal{S}} a_k$. Therefore, for any $\mathcal{S} \subseteq [K]$, $1 - \sum_{k \in \mathcal{S}} a_k \geq 0$. □

To illustrate the idea, we provide a simple example rectifier network, and convert it to a system of linear inequalities using the theorem. The rectifier network contains two hidden ReLUs ($K = 2$):

$$z = \operatorname{sgn}\Big(1 - \operatorname{rect}\big(\mathbf{w}_1 \cdot \boldsymbol{\psi} + b_1\big) - \operatorname{rect}\big(\mathbf{w}_2 \cdot \boldsymbol{\psi} + b_2\big)\Big).$$

Our theorem says that $z = 1$ if and only if the following four inequalities hold, one per subset of $[K]$:

$$\begin{cases} 1 \geq 0 & \mathcal{S} = \emptyset \\ 1 - \big(\mathbf{w}_1 \cdot \boldsymbol{\psi} + b_1\big) \geq 0 & \mathcal{S} = \{1\} \\ 1 - \big(\mathbf{w}_2 \cdot \boldsymbol{\psi} + b_2\big) \geq 0 & \mathcal{S} = \{2\} \\ 1 - \big(\mathbf{w}_1 \cdot \boldsymbol{\psi} + b_1\big) - \big(\mathbf{w}_2 \cdot \boldsymbol{\psi} + b_2\big) \geq 0 & \mathcal{S} = \{1, 2\}. \end{cases}$$

The first inequality, $1 \geq 0$, corresponding to the empty subset of $[K]$, trivially holds. The rest are just linear inequalities over $\boldsymbol{\psi}$.

In general, $[K]$ has 2^K subsets, and when $\mathcal{S}$ is the empty set, inequality (4.8) is trivially true. The rectifier network in Eq. (4.7) thus predicts $\mathbf{y}$ is a valid structure for $\mathbf{x}$, if a system of $2^K - 1$ linear inequalities are satisfied. This establishes the fact that a two-layer rectifier network of the form of Eq. (4.7) can represent a system of linear inequality constraints for a structured prediction problem via the constraint feature function $\boldsymbol{\psi}$. It is worth mentioning that the rectifier network in Eq. (4.7) cannot represent arbitrary $2^K - 1$ linear inequalities, because the $2^K - 1$ linear inequalities given by a rectifier network may not be all independent. In essence, this is an assumption on the hypothesis space for constraints, which allows for easy learning.

4.3 Learning constraints

In the previous section, we saw that both threshold and rectifier networks could represent a system of linear inequalities. We can either learn a threshold network (Eq. 4.6) to obtain a set of constraints as in (4.2), or we can learn a rectifier network (Eq. 4.7) to obtain a set of constraints as in (4.8). In both cases the number of learned constraints depends on the number of hidden units in the neural networks. As we shown in Sections 4.2.2 and 4.2.3, K hidden threshold units can represent a system of K linear inequality constraints, and K hidden rectifier units can represent a system of $2^K - 1$ linear inequality constraints. The rectifier network can express constraints more compactly. To get the same number of constraints, the rectifier network requires much fewer hidden units than threshold networks. Another advantage of using rectifier networks to learn linear constraints is that it has nontrivial gradients, which facilitates gradient-based learning.[2] Therefore we will use rectifier networks to learn constraints.

We will train the parameters $\mathbf{w}_k$'s and b_k's of the rectifier network in the supervised setting. First, we need to obtain positive and negative training examples. We assume that we have training data for a structured prediction task.

Positive examples can be directly obtained from the training data of the structured prediction problem. For each training example $(\mathbf{x}, \mathbf{y})$, we can apply constraint feature

[2]The output threshold unit in the rectifier network will not cause any trouble in practice, because it can be replaced by sigmoid function during training. Our theorem still follows, as long as we interpret $z(\mathbf{x}, \mathbf{y}) = 1$ as $\sigma(\mathbf{x}, \mathbf{y}) \geq 0.5$ and $z(\mathbf{x}, \mathbf{y}) = -1$ as $\sigma(\mathbf{x}, \mathbf{y}) < 0.5$. We can still convert the rectifier network into a system of linear inequalities even if the output unit is the sigmoid unit.

extractors to obtain positive examples of the form $(\psi(\mathbf{x}, \mathbf{y}), +1)$.

Negative examples can be generated in several ways; we use simple but effective approaches. We can slightly perturb a structure $\mathbf{y}$ in a training example $(\mathbf{x}, \mathbf{y})$ to obtain a structure $\mathbf{y}'$. Applying the constraint feature extractor to it gives a negative example $(\psi(\mathbf{x}, \mathbf{y}'), -1)$. We also need to ensure that $\psi(\mathbf{x}, \mathbf{y}')$ is indeed different from any positive example. Another approach is to perturb the feature vector $\psi(\mathbf{x}, \mathbf{y})$ directly, instead of perturbing the structure $\mathbf{y}$. In the experiment sections, we will use both methods to generate negative examples.

In our experiments, despite the simplicity of these methods for generating negative examples, we observed performance improvements. Exploring more sophisticated methods for perturbing structures or features (e.g., using techniques explored by [95], or using adversarial learning) is a future research direction.

4.4 Synthetic ILP experiments

We first test whether constraints can be learned and whether learned constraints can help a downstream task using a synthetic experiment.

Consider the case of framing structure prediction as integer linear programming:

$$\min_{\{z_i\}} \sum_i c_i \cdot z_i. \tag{4.9}$$

$$\sum_i A_{ki} z_i \geq b_k \tag{4.10}$$

$$z_i \in \{0, 1\} \tag{4.11}$$

Here, the objective coefficient c_i denotes the cost of setting the variable z_i to 1 and the goal of prediction is to find a cost minimizing variable assignment subject to the linear constraints in (4.10). We randomly generate 100 50-dimensional ILP instances, all of which share a fixed set of random constraints. Each instance is thus defined by its objective coefficients. We reserve 30% of instances as test data. The goal is to learn the shared linear constraints in Eq. (4.10) from the training set.

We use the Gurobi Optimizer [35] to solve all the ILP instances to obtain our training examples $\{(\mathbf{c}, \mathbf{z})\}$, where $\mathbf{c}$ is the vector of objective coefficients, and $\mathbf{z}$ is the vector of the optimal solution. Each $\mathbf{z}$ in the training set is feasible, giving us positive examples $(\mathbf{z}, +1)$ for the constraint learning task.

We generate negative examples as follows: Given a positive example $(\mathbf{c}, \mathbf{z})$ from the training set, if the i^{th} coefficient $c_i > 0$ and the corresponding decision $z_i = 1$, construct $\mathbf{z}'$ from $\mathbf{z}$ by flipping the i^{th} bit in $\mathbf{z}$ from 1 to 0. Such a $\mathbf{z}'$ is a negative example for the constraint learning task because $\mathbf{z}'$ has a lower objective value than $\mathbf{z}$. Therefore, it violates at least one of the constraints in Eq. (4.10). Similarly, if $c_i < 0$ and $z_i = 0$, we can flip the i^{th} bit from 0 to 1. We perform the above steps for every coefficient of every example in the training set to generate a set of negative examples $\{(\mathbf{z}', -1)\}$.

We trained a rectifier network on these examples and converted the resulting parameters into a system of linear inequalities using Theorem 5.2. We used the learned inequalities to replace the original constraints to obtain predicted solutions. We evaluated these predicted solutions against the oracle solutions (i.e., based on the original constraints). We also computed a baseline solution for each test example by minimizing an unconstrained objective.

Table 4.1 lists four measures of the effectiveness of learned constraints. First, we want to know whether the learned rectifier network can correctly predict the synthetically generated positive and negative examples. The binary classification accuracies are listed in the first row. The second row lists the bitwise accuracies of the predicted solutions based on learned constraints, compared with the gold solutions. We see that the accuracy values of the solutions based on learned constraints are in the range from 80.2–83.5%. As a comparison, without using any constraints, the accuracy of the baseline is 56.8%. Therefore, the learned constraints can substantially improve the prediction accuracy in the down stream inference tasks. The third row lists the percentage of the predicted solutions satisfying the original constraints. Solutions based on learned constraints satisfy 69.8–74.4% of the original constraints. In contrast, the baseline solutions satisfy 55.3% of the original constraints. The last row lists the percentage of the gold solutions satisfying the learned constraints. We see that the gold solutions almost always satisfy the learned constraints.

4.5 Citation field extraction experiments

In the citation field extraction task, the input is a citation entry. The goal is to identify spans corresponding to fields such as author, title, etc. In the example below, the target labels are underlined:

Table 4.1. Effectiveness of learned constraints for the synthetic ILP experiments.

	Number of ReLUs								
	2	3	4	5	6	7	8	9	10
binary classification acc. (%)	85.1	87.3	92.1	90.3	95.0	94.3	94.1	97.7	98.0
bitwise solution acc. (%)	81.1	80.9	81.9	80.2	81.0	82.3	81.1	83.2	83.5
original constr. satisfied (%)	70.3	69.8	72.7	70.4	70.1	71.1	71.4	74.4	74.3
learned constr. satisfied (%)	95.6	98.6	98.7	99.1	97.4	98.9	99.9	99.1	99.4

[Author A . M . Turing .] [Title Computing machinery and intelligence .] [Journal Mind ,] [Volume 59 ,] [Pages 433-460 .] [Date October , 1950 .]

It has been shown that constraints are important for correctly identifying citation fields. Chang et al. [8, 9] show the utility of carefully hand-designed constraints specific to this domain. In this chapter, we show that constraints learned from the training data can improve a trained model.

4.5.1 Dataset and baseline

We use the citation dataset from [8, 9]. The dataset is split into training, development and test sets, with 300, 100 and 100 examples, respectively. We train a first-order Markov model using structured SVM on the training set with the same raw text features as in the original work [8].

4.5.2 Constraint features

We explore multiple simple constraint features $\boldsymbol{\psi}(\mathbf{x}, \mathbf{y})$ as described below.

4.5.2.1 Label existence

This feature indicates which labels exist in a citation entry. In our above example, there are six labels. Suppose there are n_l possible labels. The above example is a positive example; the feature vector of which is an n_l-dimensional binary vector. Exactly six elements, corresponding to the six labels in the example, have the value 1 and all others have the value 0. To obtain the negative examples, we iterate through every positive example and flip one bit of its feature vector. If the resulting vector is not seen in the positive set, it will be a negative example.

4.5.2.2 Label counts

Label-count features are similar to label-existence features. Instead of indicating whether a label exists using 1 or 0, label-count features records the number of times each label appears in the citation entry. The positive examples can be generated naturally from the training set. To generate negative examples, we perturb the actual labels of a positive example, as opposed to its feature vector. We then extract the label counts feature from the perturbed example, and treat it as negative if it has not seen before in the positive set.

4.5.2.3 Bigram labels

This feature considers each pair of adjacent labels in the text. From left to right, the above example will give us feature vectors like (`Author`, `Author`), (`Author`, `Title`), (`Title`, `Title`), ..., (`Date`, `Date`). We then use one-hot encoding to represent these features, which is the input vector to the rectifier network. All these feature vectors are labeled as positve (+1) by the rectifier network, since they are generated from the training set. To generate negative examples for bigram-label features, we generate all positive examples from the training set, then enumerate all possible pair of labels and select those that were not seen in the positive examples.

4.5.2.4 Trigram labels

This feature is similar to the bigram labels. From the training set, we generate positive examples, e.g., (`Author`, `Author`, `Author`), (`Author`, `Author`, `Title`) etc., and convert them into one-hot encodings. For negative examples, we enumerate all possible trigram labels, and select those trigrams as negative if two conditions are met: (a) the trigram is not seen in the positive set; and (b) a bigram contained in it is seen in the training set. The intuition is that we want negative examples to be almost feasible.

4.5.2.5 Part-of-speech

For a fixed window size, we extract part-of-speech tags and the corresponding labels, and use the combination as our constraint features. For example, with window size two, we get indicators for ($\texttt{tag}_{i-1}$, $\texttt{tag}_i$, $\texttt{label}_{i-1}$, $\texttt{label}_i$) for the i^{th} token in the sentence, where `tag` and `label` refer to part-of-speech tag and citation field label, respectively. For negative examples, we enumerate all four-tuples as above, and select it as negative if the four-tuple

is not seen in the positive set, but both ($\texttt{tag}_{i-1}$, $\texttt{tag}_i$) and ($\texttt{label}_{i-1}$, $\texttt{label}_i$) are seen in the training set.

4.5.2.6 Punctuation

The punctuation feature is similar to the part-of-speech feature. Instead of the POS tag, we use an indicator for whether the current token is punctuation.

4.5.3 Experiments and results

For each constraint feature mentioned previously, we trained a rectifier network with 10 ReLUs in the hidden layer. We then use theorem 5.2 to convert the resulting network to a system of $2^{10} - 1$, or 1023 linear inequalities. We used beam search (beam size 50) to combine the learned inequalities with the original sequence model to predict on the test set. States in the search space correspond to partial assignments to a prefix of the sequence. Each step we predict the label for the next token in the sequence. The pretrained sequence model (i.e., the baseline) ranks search nodes based on transition and emission scores, and the learned inequality prunes the search space accordingly.[3] Table 4.2 shows the token level accuracies of various methods. We can see that all versions of constrained search perform better than the baselines, indicating that the learned constraints are effective in the citation field extraction task. Furthermore, different constraints learned with different features can be combined. We observe that combining different constraint features generally improves accuracy.

It is worth pointing out that the label existence and label counts features are global in nature and cannot be directly used to train a sequence model. Even if some constraint features can be used in training the original model, it is still beneficial to learn constraints from them. For example, the bigram label feature is captured in the original first order model, but adding constraints learned from them still improves performance. As another test, we trained a model with POS features, which also contains punctuation information. This model achieves 91.8% accuracy. Adding constraints learned with POS improves the accuracy to 92.6%; adding constraints learned with punctuation features further improves

[3]The label-existence and label-counts features are global, which means pruning by learned inequalities is possible only at the very last step of the search process. For all the other four features introduced in the text, pruning can be achieved in every step of the search process.

Table 4.2. Token level accuracies (in percentage) of baseline models and constrained-search models, for the citation field extraction task. **Exact** is our trained first-order Markov model. It uses exact inference (dynamic programming) for prediction. **Search** is our search baseline; it uses the same model as **Exact**, but with beam search for inexact inference. **L.E., L.C., B.L., T.L., POS,** and **Punc.** use search with different constraint features: label existence, label counts, bigram labels, trigram labels, part-of-speech, and punctuation features. **C1** to **C3** are search with combined constraints. **C1** combines **L.E.** and **T.L.**. **C2** combines **L.E.**, **T.L.**, and **POS**. Finally **C3** combines all constraints.

Baselines		**Search with learned constraints**						**Combine constraints**		
Exact	**Search**	**L.E.**	**L.C.**	**B.L.**	**T.L.**	**POS**	**Punc.**	**C1**	**C2**	**C3**
86.2	87.3	88.0	87.7	87.9	88.1	89.8	90.2	88.6	90.1	90.6

it to 93.8%.

We also observed that our method for learning constraints is robust to the choice of the number of hidden ReLUs. As shown in Table 4.3, as long as the number of units is not too small (≥ 5), the accuracy does not change much. The accuracy initially increases with increasing number of ReLUs, and then becomes saturate very quickly. This table uses the punctuation feature. We observed similar behavior for other constraint features as well. Since the number of constraints learned is exponential in the number of hidden units, a large number of ReLUs will result in a large number of redundant constraints. These results show that learning redundant constraints will not help performance, and will not hurt either. We can use validation data to select the optimal number of ReLUs.

Note that carefully hand-designing constraints can achieve higher accuracy than the learned constraints. Chang et al. [8] report an accuracy of 92.5% with constraints specifically constructed for this domain. In contrast, our method of automatically learning constraints uses only general constraint features and does not rely on domain knowledge. Therefore, our method is particularly suited to tasks where little is known about the un-

Table 4.3. Token level accuracies as a function of number of ReLUs in the hidden layer of the rectifier network. The constraint feature used is the punctuation feature.

Number of hidden ReLUs			
$n = 3$	$n = 5$	$n = 8$	$n = 10$
82.6%	90.1%	90.3%	90.2%

derlying constraints. Interpreting such learned constraints is a direction of future research.

4.6 Entity and relation extraction experiments

In the task of entity and relation extraction, we are given a sentence with entity candidates. We seek to determine the type of each candidate, as in the following:

> [Organization Google] is headquartered in [Location Mountain View, California.]

We also want to determine directed relations between the entities. In the above example, the relation from "Google" to "Mountain View, California" is `OrgBasedIn`, and the opposite direction is labeled `NoRel`, indicating there is no relation. Unlike the citation task, which is a sequence labeling problem, this task requires predicting a directed graph and represents a typical structured prediction problem — we cannot make isolated entity and relation predictions.

4.6.1 Dataset and baseline

We use the dataset from [85]. It contains 1441 sentences with labeled entities and relations. As shown in the example in Chapter 1, there are three possible entity labels: `Person`, `Location` and `Organization` and five possible relations: `Kill`, `LiveIn`, `WorkFor`, `LocatedAt` and `OrgBasedIn`. Additionally, there is a special entity label `NoEnt` meaning a text span is not an entity, and a special relation label `NoRel` indicating that two spans are unrelated.

We used 70% of the data for training and the remaining 30% for evaluation. We trained our baseline model using the integer linear program (ILP) formulation with the same set of features as in [85]. The baseline system includes manually designed constraints from the original paper. Table 4.4 lists the designed constraints used in the entity and relation extraction experiments. There are fifteen constraints, three for each relation type. For example, the last row in Table 4.4 means that the relation `OrgBasedIn` must have an `Organization` as its source entity and a `Location` as its target entity, and the relation in the opposite direction must be `NoRel`.

Table 4.4. Designed constraints used in the entity and relation extraction experiments

Antecedents	Consequents		
If the relation is	Source must be	Target must be	Reversed relation must be
`Kill`	`Person`	`Person`	`NoRel`
`LiveIn`	`Person`	`Location`	`NoRel`
`WorkFor`	`Person`	`Organization`	`NoRel`
`LocatedAt`	`Location`	`Location`	`NoRel`
`OrgBasedIn`	`Organization`	`Location`	`NoRel`

4.6.2 Constraint features

We used three constraint features for this task.

4.6.2.1 Source-relation indicator

This feature looks at a given relation label and the label of its source entity. It has the form of ($\texttt{label}_{\text{source}}$, $\texttt{label}_{\text{relation}}$). Our example sentence will contribute the following two feature vectors (`Organization`, `OrgBasedIn`) and (`Location`,`NoRel`), both corresponding to postive examples. One-hot encoding can again be used to convert these features to real valued vectors. The negative examples contains all possible pairs of ($\texttt{label}_{\text{source}}$, $\texttt{label}_{\text{relation}}$), which do not appear in the positive example set.

4.6.2.2 Relation-target indicator

This feature looks at a given relation label and its target label. It has the form of ($\texttt{label}_{\text{relatioin}}$, $\texttt{label}_{\text{target}}$). Our example sentence will contribute the following two feature vectors (`OrgBasedIn`, `Location`) and (`NoRel`,`Organization`), both corresponding to positive examples. The negative examples contains all possible pairs of ($\texttt{label}_{\text{relation}}$, $\texttt{label}_{\text{target}}$), which do not appear in the positive example set.

4.6.2.3 Relation-relation indicator

The relation-relation feature looks at a pair of entities and focuses on the two relation labels between them, one in each direction. Therefore, our running example will give us two positive examples with features (`OrgBasedIn`, `NoRel`) and (`NoRel`,`OrgBasedIn`). The negative examples contain any pair of relation labels that is not seen in the positive example set.

4.6.3 Experiments and results

We compared the performance of two ILP-based models, both trained in the presence of constraints with a structured SVM. One model was trained with manually designed constraints and the other used learned constraints. These models are compared in Table 4.5. In all performance metrics, the model *trained* with the learned constraints outperforms the model trained with designed constraints.

We have shown that $2^K - 1$ linear inequality constraints are learned using a rectifier network with K hidden units. In the entity and relation extraction experiments, we use two hidden units to learn three constraints from the source-relation indicator features. The three learned constraints are listed in Table 4.6. A given pair of source label and relation label satisfies the constraint if the sum of the corresponding coefficients and the bias term is greater than or equal to zero. For example, one learned constraint (first row in Table 4.6) using the source-relation indicator features is

$$\begin{aligned} &-1.98x_1 + 3.53x_2 - 1.90x_3 + 0.11x_4 \\ &\quad + 2.66x_5 - 2.84x_6 - 2.84x_7 - 2.84x_8 + 2.58x_9 + 0.43x_{10} + 0.32 \geq 0, \end{aligned} \tag{4.12}$$

where x_1 through x_{10} are indicators for labels `NoEnt`, `Person`, `Location`, `Organization`, `NoRel`, `Kill`, `LiveIn`, `WorkFor`, `LocatedAt`, and `OrgBasedIn`, respectively. This constraint disallows a relation labeled as `Kill` having a source entity labeled as `Location`, because $-1.90 - 2.84 + 0.32 < 0$. Therefore, the constraint "`Location` cannot `Kill`" is captured in (4.12). In fact, it is straightforward to verify that the inequality in (4.12) captures many more constraints such as "`NoEnt` cannot `LiveIn`," "`Location` cannot `LiveIn`," and "`Organization` cannot `WorkFor`," etc. A general method for interpreting learned constraints is a direction of future research.

Table 4.5. Comparison of performance on the entity and relation extraction task, between two ILP models, one trained with designed constraints (**Designed**) and one with learned constraints (**Learned**).

Performance Metric	**Designed**	**Learned**
entity F-1	82.7%	85.9%
relation F-1	48.2%	55.5%

Table 4.6. Linear constraint coefficients learned from the source-relation indicator features

Source Labels				Relation Labels						
NoEnt	Per.	Loc.	Org.	NoRel	Kill	Live	Work	Located	Based	Bias
-1.98	3.53	-1.90	0.11	2.66	-2.84	-2.84	-2.84	2.58	0.43	0.32
-1.61	-1.48	3.50	0.92	1.15	1.02	1.02	1.02	-3.96	-1.38	1.46
-3.59	2.04	1.60	1.03	3.81	-1.82	-1.82	-1.82	-1.38	-0.95	0.78

We enumerated all possible pairs of source label and relation label and found that the learned constraints always agree with the designed constraints in the following sense: whenever a pair of source label and relation label satisfies the designed constraints, it also satisfies all three learned constraints, and whenever a pair of source label and relation label is disallowed by the designed constraints, it violates at least one of the learned constraints. Therefore, our method of constraint learning exactly recovers the designed constraints.

We also use two hidden units to learn three constraints from the relation-target indicator features, and one hidden unit to learn one constraint from the relation-relation indicator features. The learned constraints are listed in Table 4.7 and Table 4.8. Again we verify that the learned constraints exactly recover the designed constraints in all cases.

Table 4.7. Linear constraint coefficients learned from the relation-target indicator features

Relation Labels						Target Labels				
NoRel	Kill	Live	Work	Located	Based	NoEnt	Per.	Loc.	Org.	Bias
2.68	-3.17	-0.55	2.68	-0.55	-0.55	-1.58	3.15	0.53	-2.70	1.02
2.72	2.42	-1.39	-2.55	-1.39	-1.39	-2.51	-2.27	1.54	2.31	0.85
5.40	-0.74	-1.94	0.13	-1.94	-1.94	-4.10	0.88	2.08	-0.39	0.86

Table 4.8. Linear constraint coefficients learned from the relation-relation indicator features. The order of the relation labels is `NoRel`, `Kill`, `LiveIn`, `WorkFor`, `LocatedAt`, and `OrgBasedIn`.

Forward Relation Labels						Backward Relation Labels						Bias
4.95	-1.65	-1.65	-1.65	-1.65	-1.65	5.06	-1.53	-1.53	-1.53	-1.53	-1.53	-2.41

4.7 Conclusions

In this chapter, we presented a systematic way of discovering constraints as linear inequalities for structured prediction problems. Expressing constraints as linear inequalities is particularly suited for structured prediction problems, since they can be used directly with a variety of inference frameworks. The learned inequalities can also be combined with hand-designed constraints, if any, so that both domain knowledge and patterns from data can be leveraged to obtain a structured model.

The proposed approach is built upon a novel transformation from two-layer rectifier networks to linear inequality constraints and does not rely on domain expertise for any specific problem. Instead, it only uses general constraint features as inputs to rectifier networks. Our approach is especially useful for tasks where designing constraints manually is hard. The learned constraints can be used for structured prediction problems in two ways: (1) combining them with an existing model to improve prediction performance, or (2) incorporating them into the training process to train a better model. We demonstrated the effectiveness of our approach on one synthetic ILP experiment and two information extraction tasks, where learned constraints substantially help prediction quality.

An important future research direction is to study the interpretability of the learned constraints. As we saw in Section 4.6.3, the learned linear inequality constraints involve all constraint features, which make it hard to interpret the learned constraints directly. It is desirable to learn sparse constraints to facilitate explanation of the learned constraints. One possible way of achieving sparse linear inequality is to use sparse regularizers during the process of neural network learning.

It is worth mentioning that our methods in this chapter are orthogonal to the advantages provided by the neural network based models. The neural models either completely ignore the output dependencies, as in an RNN/BERT based models [22, 100, 104], or treat the neural network as a way to get unary potentials and rely on standard structured prediction for output interactions [24].

CHAPTER 5

LEARNING TO SPEED UP STRUCTURED OUTPUT PREDICTION

We see in Chapter 4 that suitably designed constraints are crucial in successful modeling a structured prediction problem, and they can be learned using neural networks from training data. However, in the presence of complicated constraints, predicting structured outputs can be computationally onerous due to the combinatorially large output spaces and the discrete nature of the output spaces. In this chapter, we focus on reducing the prediction time of a trained black-box structured classifier without losing accuracy. To do so, we train a *speedup* classifier that learns to mimic a black-box classifier under the learning-to-search approach. As the structured classifier predicts more examples, the speedup classifier will operate as a learned heuristic to guide search to favorable regions of the output space. We present a mistake bound for the speedup classifier and identify inference situations where it can independently make correct judgments without input features. We evaluate our method on the task of entity and relation extraction and show that the speedup classifier outperforms even greedy search in terms of speed without loss of accuracy.

5.1 Introduction

Many natural language processing (NLP) and computer vision problems necessitate predicting structured outputs such as labeled sequences, trees or general graphs [68, 94]. Such tasks require modeling both input-output relationships and the interactions between predicted outputs to capture correlations. Across the various structured prediction formulations [9, 49, 101], prediction requires solving inference problems by searching for score-maximizing output structures. The search space for inference is typically large (e.g., all parse trees), and grows with input size. Exhaustive search can be prohibitive and standard alternatives are either (a) perform exact inference with a large computational cost or

(b) approximate inference to sacrifice accuracy in favor of time.

In this paper, we focus on the computational cost of inference. We argue that naturally occurring problems have remarkable regularities across both inputs and outputs, and traditional formulations of inference ignore them. For example, parsing an n-word sentence will cost a standard head-driven lexical parser $O(n^5)$ time. Current practice in NLP is to treat each new sentence as a new discrete optimization problem and pay the computational price each time. However, this practice is not only expensive, but also wasteful! We ignore the fact that slight changes to inputs often do not change the output or even the sequence of steps taken to produce it. Moreover, not all outputs are linguistically meaningful structures; as we make more predictions, we should be able to *learn* to prune the output space.

The motivating question that drives our work is the following: *Can we design inference schemes that learn to make a trained structured predictor faster without sacrificing output quality?* After training, the structured classifier can be thought as a black-box. Typically, once deployed, it is never modified over its lifetime of classifying new examples. Subsequently, we can view each prediction of the black-box classifier as an opportunity to learn how to navigate the output space more efficiently. Thus, if the classifier sees a previously encountered situation, it could make some decisions without needless computations.

We formalize this intuition by considering the trained models as solving arbitrary integer linear programs (ILPs) for combinatorial inference. We train a second, inexpensive *speedup* classifier that acts as a heuristic for a search-based inference algorithm that mimics the more expensive black-box classifier. The speedup heuristic is a function that learns regularities among *predicted* structures. We present a mistake bound algorithm that, over the classifier's lifetime, learns to navigate the feasible regions of the ILPs. By doing so, we can achieve a reduction in inference time.

We further identify inference situations where the learned speedup heuristic alone can correctly label parts of the outputs without computing the corresponding input features. In such situations, the search algorithm can safely *ignore* parts of inputs if the corresponding outputs can be decided based on the substructures constructed so far. Seen this way, the speedup classifier can be seen as a statistical cache of past decisions made by the black-box classifier.

We instantiate our strategy to the task of predicting entities and relations from sentences. Using an ILP based black-box classifier, we show that the trained speedup classifier mimics the reference inference algorithm to obtain improvements in running time, and also recovers its accuracy. Indeed, by learning to ignore input components when they will not change the prediction, we show that learned search strategy outperforms even greedy search in terms of speed.

To summarize, the main contribution of this paper is the formalization of the problem of learning to make structured output classifiers faster without sacrificing accuracy. We develop a learning-to-search framework to train a speedup classifier with a mistake-bound guarantee and a sufficient condition to safely avoid computing input-based features. We show empirically on an entity-relation extraction task that we can learn a speedup classifier that is (a) faster than both the state-of-the-art Gurobi optimizer and greedy search, and (b) does not incur a loss in output quality.

5.2 Notation and preliminaries

First, we will define the notation used in this paper with a running example that requires identifying entity types and their relationships in text. The input to the problem consists of sentences such as the following:

Colin went back home in Ordon Village.

These inputs are typically preprocessed — here, we are given spans of text (underlined) corresponding to entities. We will denote such preprocessed inputs to the structured prediction problem as $\mathbf{x}$.

We seek to produce a structure $\mathbf{y} \in \mathcal{Y}_\mathbf{x}$ (e.g., labeled trees, graphs) associated with these inputs. Here, $\mathcal{Y}_\mathbf{x}$ is the set of all possible output structures for the input $\mathbf{x}$. In the example problem, our goal is to assign types to the entities and also label the relationships between them. Suppose our task has three types of entities: `person`, `location`, and `organization`. A pair of entities can participate in one of five possible directed relations: `Kill`, `LiveIn`, `WorkFor`, `LocatedAt`, and `OrgBasedIn`. Additionally, there is a special entity label `NoEnt` meaning a text span is not an entity, and a special relation label `NoRel` indicating that two spans are unrelated. Figure 5.1 shows a plausible structure for the example sentence as per this scheme.

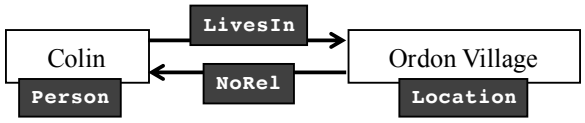

Figure 5.1. An example of the entities and relations task. The nodes are entity candidates, and directed edges indicate relations. The labels in typewriter font are the decisions that we need to make.

A standard way to model the prediction problem requires learning a model that scores all structures in $\mathcal{Y}_{\mathbf{x}}$ and searching for the score-maximizing structure. Linear models are commonly used as scoring functions, and require a feature vector characterizing input-output relationships $\Phi(\mathbf{x}, \mathbf{y})$. We will represent the model by a weight vector α. Every structure $\mathbf{y}$ associated with an input $\mathbf{x}$ is scored as the dot product $\alpha \cdot \Phi(\mathbf{x}, \mathbf{y})$. The goal of prediction is to find the structure $\mathbf{y}^*$ that maximizes this score. That is,

$$\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} \alpha \cdot \Phi(\mathbf{x}, \mathbf{y}). \tag{5.1}$$

Learning involves using training data to find the best weight vector α.

In general, the output structure $\mathbf{y}$ is a set of K categorical variables $\{y^1, y^2, \cdots, y^K\}$, each of which can take a value from a predefined set of n labels. That is, each $y^k \in \mathbf{y}$ takes a value from $\{l_1, l_2, \cdots, l_n\}$.[1] In our running example, the inference variables correspond to the four decisions that define the structure: the labels for the two entities, and the relations in each direction. The feature function Φ decomposes into a sum of features over each y^k, each denoted by Φ_k, giving us the inference problem:

$$\mathbf{y}^* = \arg\max_{\mathbf{y} \in \mathcal{Y}_{\mathbf{x}}} \sum_{k=1}^{K} \alpha \cdot \Phi_k\left(\mathbf{x}, y^k\right). \tag{5.2}$$

The dependencies between the y^k's specify the nature of the output space. Determining each y^k in isolation greedily does not typically represent a viable inference strategy because constraints connecting the variables are ignored. In this spirit, the problem of finding the best structure can be viewed as a combinatorial optimization problem.

In this paper, we consider the scenario in which we have already trained a model α. We focus on solving the inference problem (i.e.,Eq. (5.2)*) efficiently.* We conjecture that it should be

[1]We make this choice for simplicity of notation. In general, K depends on the size of the input $\mathbf{x}$, and categorical variables may take values from different label sets.

possible to observe a black-box inference algorithm over its lifetime to learn to predict faster without losing accuracy.

5.2.1 Black-box inference mechanisms

One common way to solve inference is by designing efficient dynamic programming algorithms that exploit problem structure. While effective, this approach is limited to special cases where the problem admits efficient decoding, thus placing restrictions on factorization and feature design.

In this paper, we seek to reason about the problem of predicting structures in the general case. Since inference is essentially a combinatorial optimization problem, without loss of generality, we can represent any inference problem as an integer linear programming (ILP) instance [90]. To represent the inference task in Eq. (5.2) as an ILP instance, we will define indicator variables of the form $z_i^k \in \{0, 1\}$, which stands for the decision that the categorical variable y^k is assigned the i^{th} label among the n labels. That is, $z_i^k = 1$ if $y^k = l_i$, and 0 otherwise. Using this notation, we can write the cost of any structure $\mathbf{y}$ in terms of the indicators as

$$\sum_{k=1}^{K} \sum_{i=1}^{n} c_i^k z_i^k. \tag{5.3}$$

Here, c_i^k is a stand in for $-\alpha \cdot \Phi_k(\mathbf{x}, l_i)$, namely the cost (negative score) associated with this decision.[2] In our example, suppose the first categorical variable y^1 corresponds to the entity *Colin*, and it has possible labels $\{\texttt{person}, \texttt{location}, \dots\}$. Then, assigning `person` to *Colin* would correspond to setting $z_1^1 = 1$, and $z_i^1 = 0$ for all $i \neq 1$. Using the labels enumerated in Section 5.2, there will be 20 indicators for the four categorical decisions.

Of course, arbitrary assignments to the indicators are not allowed. We can define the set of feasible structures using linear constraints. Each categorical variable can take exactly one label, which can be expressed via:

$$\sum_{i=1}^{n} z_i^k = 1, \quad \text{for all } k. \tag{5.4}$$

In addition, we can define the set of valid structures $\mathcal{Y}_\mathbf{x}$ using a collection of m linear constraints; the j^{th} one of which can be written as

[2] The negation defines an equivalent minimization problem and makes subsequent description of the search framework easier.

$$\sum_{k=1}^{K}\sum_{i=1}^{n} A_{ji}^{k} z_i^k = b_j, \quad \text{for all } j. \tag{5.5}$$

These *structural* constraints characterize the interactions between the categorical variables. For example, if a directed edge in our running example is labeled as `LiveIn`, then, its source and target must be a `person` and a `location` respectively. While Eq.(5.5) only shows equality constraints, in practice, inequality constraints can also be included.

The inference problem in Eq. (5.2) is equivalent to the problem of minimizing the objective in Eq. (5.3) over the 0-1 indicator variables subject to the constraints in Eqs. (5.4) and (5.5). It is worth pointing out that there are multiple ways of converting the inference problem in Eq. (5.2) to an ILP instance. The proposed one, which is a systematic binarization of discrete output decisions, is the most natural one. The advantage of this binarization method is that the coefficient for each indicator variable can be viewed as the numeric score of corresponding assignment decision.

We should note the difference between the ability to write an inference problem as an ILP instance and actually *solving* it as one. The former gives us the ability to reason about inference in general, and perhaps using other methods (such as Lagrangian relaxation [51]) for inference. However, solving problems with industrial strength ILP solvers such as the Gurobi solver [35] is competitive with other approaches in terms of inference time, even though they may not directly exploit problem structure.

In this work, we use the general structure of the ILP inference formulation to develop the theory for speeding up inference. In addition, because of its general applicability and fast inference speed, we use the Gurobi ILP solver as our black-box classifier, and learn a speedup heuristic to make an even faster inference.

5.2.2 Inference as search

Directly applying the black-box solver for the large output spaces may be impractical. An alternative general-purpose strategy for inference involves framing the maximization in Eq. (5.2) as a graph search problem.

Following [89, 110], a general graph search problem requires defining an initial search node I, a successor function $s(\cdot)$, and a goal test. The successor function $s(\cdot)$ maps a search node to its successors. The goal test determines whether a node is a goal node. Usually,

each search step is associated with a cost function, and we seek to find a goal node with the least total cost.

We can define the search problem corresponding to inference as follows. We will denote a generic search node in the graph as v, which corresponds to a set of partially assigned categorical variables. Specifically, we will define the search node v as a set of pairs $\{(k, i)\}$; each element of which specifies that the variable y^k is assigned the i^{th} label. The initial search node I is the empty set since none of the variables has been assigned when the search begins. For a node v, its successors $s(v)$ is a set of nodes, each containing one more assigned variable than v. A node is a goal node if all variables y^k's have been assigned. The size of any goal node is K, the number of categorical variables.

In our running example, at the start of the search, we may choose to assign the first label l_1 (`person`) to the variable y^1 – the entity *Colin* – leading us to the successor $\{(1, 1)\}$. Every search node specifies a partial or a full assignment to all the entities and relations. The goal test checks if we arrive at a full assignment: i.e., all the entity and relation candidates have been assigned a label.

Note that the goal test does not test the quality of the node; it merely tests whether the search process is finished. The quality of the goal node is determined by the path cost from the initial node to the goal node, which is the accumulated cost of each step along the way. The step cost for assigning label l_i to a variable y^k is the same c_i^k we defined for the ILP objective in Eq. (5.3). Finding the shortest path in such a search space is equivalent to the original ILP problem without the structural constraints in Eq. (5.5). The unique-label constraints in Eq. (5.4) are automatically satisfied by our formulation of the search process.

Indeed, solving inference without the constraints in Eq.(5.5) is trivial. For each categorical variable y^k, we can pick the label l_i that has the lowest value of c_i^k. This gives us two possible options for solving inference as search: we can (a) ignore the constraints that make inference slow to greedily predict all the labels, or (b) enforce constraints at each step of the search, and only consider search nodes that satisfy all constraints. The first option is fast, but can give us invalid outputs. For example, we might get a structure that mandates that the `person` *Colin* lives in a `person` called *Ordon Village*. The second option will give us structurally valid outputs, but can be prohibitively slow.

Various graph search algorithms can be used for performing inference. For efficiency,

we can use beam search with a fixed beam width b. When search begins the beam B_0 contains only the initial node $B_0 = [I]$. Following [14, 110], we define the function **BreadthExpand**, which takes the beam B_t at step t and generates the candidates C_{t+1} for the next beam:

$$\begin{aligned} C_{t+1} &= \textbf{BreadthExpand}(B_t) \\ &= \cup_{v \in B_t} s(v). \end{aligned}$$

The next beam is given by $B_{t+1} = \textbf{Filter}(C_{t+1})$, where **Filter** takes top b nodes according to some priority function $p(v)$. In the simplest case, the priority of a node v is the total path cost of reaching that node. More generally, the priority function can be informed not only by the path cost, but also by a heuristic function as in the popular A^* algorithm.

5.3 Speeding up structured prediction

In the previous section, we saw that using a black-box ILP solver may be slower than greedy search which ignores constraints, but produces valid outputs. However, over its lifetime, a trained classifier predicts structures for a large number of inputs. While the number of unique inputs (e.g., sentences) may be large, the number of unique structures that actually occur among the predictions is not only finite, but also small. This observation was exploited by [48, 98] for amortizing inference costs.

In this paper, we are driven by the need for an inference algorithm that learns regularities across outputs to become faster at producing structurally valid outputs. In order to do so, we will develop an inference-as-search scheme that inherits the speed of greedy search, but learns to produce structurally valid outputs. Before developing the algorithmic aspects of such an inference scheme, let us first see a proof-of-concept for such a scheme.

5.3.1 Heuristics for structural validity

Our goal is to incorporate the structural constraints from Eq. (5.5) as a heuristic for greedy or beam search. To do so, at each step during the search, we need to estimate how likely an assignment can lead to a constraint violation. This information can be characterized by using a heuristic function $h(v)$, which will be used to evaluate a node v during the search.

The dual form the ILP in Eqs. (5.3) to (5.5) help justify the idea of capturing constraint information using a heuristic function. We treat the unique label constraints in Eq. (5.4) as defining the domain in which each 0-1 variable z_i^k lives, and the only real constraints are given by Eq. (5.5).

Let u_j represent the dual variable for the j^{th} constraint. Thus, we obtain the Lagrangian[3]

$$\begin{aligned} L(z,u) &= \sum_{k=1}^{K}\sum_{i=1}^{n} c_i^k z_i^k - \sum_{j=1}^{m} u_j \left(\sum_{k=1}^{K}\sum_{i=1}^{n} A_{ji}^k z_i^k - b_j \right) \\ &= \sum_{k,i} \left(c_i^k - \sum_j u_j A_{ji}^k \right) z_i^k + \sum_j b_j u_j. \end{aligned}$$

The dual function $\theta(u) = \min_z L(z,u)$, where the minimization is over the domain of the z variables.

Denote $u^* = \arg\max \theta(u)$ as the solution to the dual problem. In the case of zero duality gap, the theory of Lagrangian relaxation [51] tells us that solving the following relaxed minimization problem will solve the original ILP:

$$\min \sum_{k,i} \left(c_i^k - \sum_j u_j^* A_{ji}^k \right) z_i^k \tag{5.6}$$

$$\sum_i z_i^k = 1, \quad \text{for all } k \tag{5.7}$$

$$z_i^k \in \{0,1\}, \quad \text{for all } k,i. \tag{5.8}$$

This new optimization problem does not have any structural constraints and can be solved greedily for each k if we know the optimal dual variables u^*.

To formulate the minimization in Eqs (5.6) to (5.8) as a search problem, we define the priority function $p(v)$ for ranking the nodes as $p(v) = g(v) + h^*(v)$, where the path cost $g(v)$ and heuristic function $h^*(v)$ are given by

$$g(v) = \sum_{(k,i)\in v} c_i^k, \tag{5.9}$$

$$h^*(v) = -\sum_{(k,i)\in v} \sum_j A_{ji}^k u_j^*(\mathbf{x}). \tag{5.10}$$

Since Eq. (5.6) is a minimization problem, smaller priority value $p(v)$ means higher ranking during search. Note that even though heuristic function defined in this way is not always

[3]We omit the ranges of the summation indices i, j, and k hereafter.

admissible, greedy search with ranking function $p(v)$ will lead to the exact solution of Eqs. (5.6) to (5.8). In practice, however, we do not have the optimal values for the dual variables u^*. Indeed, when Lagrangian relaxation is used for inference, the optimal dual variables are computed using subgradient optimization for each example because their value depends on the original input via the c's.

Instead of performing expensive gradient-based optimization for *every* input instance, we will approximate the heuristic function as a classifier that learns to prioritize structurally valid outputs. In this paper, we use a linear model based on a weight vector $\mathbf{w}$ to approximate the heuristic as

$$h(v) = -\mathbf{w} \cdot \boldsymbol{\phi}(v). \tag{5.11}$$

For an appropriate choice of node features $\boldsymbol{\phi}(v)$, the heuristic $h(v)$ in Eq.(5.12) is indeed a linear function.[4] In other words, there exists a linear heuristic function that can guide graph search towards creating structurally valid outputs.

In this setting, the priority function $p(v)$ for each node is determined by two components: the path cost $g(v)$ from the initial node to the current node, and the learned heuristic cost $h(v)$, which is an estimate of how good the current node is. Because the purpose of the heuristic is to help improve inference speed, we call $\boldsymbol{\phi}(v)$ speedup features. The speedup features can be different from the original model features in Eq. (5.2). In particular, it can include features for partial assignments made so far, which were not available in the original model features. In this setting, the goal of speedup learning is to find a suitable weight vector $\mathbf{w}$ over the black-box classifier's lifetime.

5.3.2 Linear heuristic function

In Section 5.3.1 we show that greedy search combined with the priority function $p(v) = g(v) + h^*(v)$ will lead to the exact solution of the original ILP (Eqs. (5.3) to (5.5)), under the condition that there is no duality gap. For convenience, we repeat the definition of the optimal heuristic function $h^*(v)$ here:

$$h^*(v) = -\sum_{(k,i) \in v} \sum_j A^k_{ji} u^*_j(\mathbf{x}) \tag{5.12}$$

[4] See the next section for an elaboration.

In this section, we show that the heuristic function in Eq. (5.12) can be written as a linear function of the form $-\mathbf{w} \cdot \phi(v)$.

First, let us recap the meanings and ranges of indices k, i, and j in Eq. (5.12):

- index k (from 1 to K): the k^{th} categorical variable.
- index i (from 1 to n): the i^{th} label.
- index j (from 1 to m): the j^{th} constraint.

Second, recall that a search node v is just a set of pairs $\{(k, i)\}$; each element of which specifies that the variable y^k is assigned the i^{th} label.

Now we can define a $K \times n \times m$ dimensional feature vector $\phi(v)$; the component of which is labeled by a tuple of indices (k, i, j). Let the $(k, i, j)^{\text{th}}$ component of the feature vector be

$$\phi_{kij}(v) = \begin{cases} u_j^*(\mathbf{x}), & \text{if } (k, i) \in v \\ 0, & \text{otherwise.} \end{cases} \tag{5.13}$$

Also define the corresponding weight parameter

$$w_{kij} = A_{ji}^k. \tag{5.14}$$

Clearly the heuristic function in Eq. (5.12) is just $-\mathbf{w} \cdot \phi(v)$, where ϕ and $\mathbf{w}$ are defined in Eq. (5.13) and Eq. (5.14), respectively.

5.4 Learning the speedup classifier

In this section, we will describe a mistake-bound algorithm to learn the weight vector $\mathbf{w}$ of the speedup classifier. The design of this algorithm is influenced by learning to search algorithms such as LaSO [21, 110]. We assume that we have access to a trained black-box ILP solver called **Solve**, which can solve the structured prediction problems, and we have a large set of examples $\{\mathbf{x}_i\}_{i=1}^N$. Our goal is to use this set to train a speedup classifier to mimic the ILP solver while predicting structures for this set of examples. Subsequently, we can use the less expensive speedup influenced search procedure to replace the ILP solver.

To define the algorithm, we will need additional terminology. Given a reference solution $\mathbf{y}$, we define a node v to be *y-good*, if it can possibly lead to the reference solution. If a node v is y-good, then the already assigned variables have the same labels as in the

Algorithm 1 Learning a speedup classifier using examples $\{\mathbf{x}_i\}_{i=1}^{N}$, and a black-box Solver **Solve**.

1: Initialize the speedup weight vector $\mathbf{w} \leftarrow \mathbf{0}$
2: **for** $epoch = 1 \ldots M$ **do**
3: **for** $i = 1 \ldots N$ **do**
4: $\mathbf{y} \leftarrow \textbf{Solve}(\mathbf{x}_i)$
5: Initialize the beam $B \leftarrow [I]$
6: **while** B is y-good **and** $\hat{v}$ is not goal **do**
7: $B \leftarrow \textbf{Filter}(\textbf{BreadthExpand}(B))$
8: **end while**
9: **if** B is not $\mathbf{y}$-good **then**
10: $v^* \leftarrow \textbf{SetGood}(\hat{v})$
11: $\mathbf{w} \leftarrow \mathbf{w} + \phi(v^*) - \frac{1}{|B|}\sum_{v \in B} \phi(v)$
12: **else if** $\hat{v}$ is not y-good **then**
13: $v^* \leftarrow \textbf{SetGood}(\hat{v})$
14: $\mathbf{w} \leftarrow \mathbf{w} + \phi(v^*) - \phi(\hat{v})$
15: **end if**
16: **end for**
17: **end for**

reference solution. We define a beam B is y-good if it contains at least one y-good node to represent the notion that search is still viable. We denote the first element (the highest ranked) in a beam by $\hat{v}$. Finally, we define an operator **SetGood**, which takes a node that is not y-good, and return its corresponding y-good node by fixing the incorrect assignments according to the reference solution. The unassigned variables are still left unassigned by the **SetGood** operator.

The *speedup-learning* algorithm is listed as Algorithm 1. It begins by initializing the weight $\mathbf{w}$ to the zero vector. We iterate over the examples for M epochs. For each example $\mathbf{x}_i$, we first solve inference using the ILP solver to obtain the reference structure $\mathbf{y}$ (line 4). Next, a breadth-expand search is performed (lines 5-8). Every time the beam B is updated, we check if the beam contains at least one y-good node that can possibly lead to the reference solution $\mathbf{y}$. The search terminates if the beam is not y-good, or if the highest ranking node $\hat{v}$ is a goal. If the beam is not y-good, we compute the corresponding y-good node v^* from $\hat{v}$, and perform a perceptron style update to the speedup weights (line 9-11). In other words, we update the weight vector by adding the feature vector of $\phi(v^*)$, and subtracting the average feature vector of all the nodes in the beam. Otherwise $\hat{v}$ must be a goal node. We then check if $\hat{v}$ agrees with the reference solution (lines 12-15). If not, we

perform a similar weight update, by adding the feature vector of $\phi(v^*)$, and subtracting $\phi(\hat{v})$.

5.4.1 Mistake bound

Next, we show that the Algorithm 1 has a mistake bound. Let R_ϕ be a positive constant such that for every pair of nodes (v, v'), we have $\|\phi(v) - \phi(v')\| \leq R_\phi$. Let R_g be a positive constant such that for every pair of search nodes (v, v'), we have $|g(v) - g(v')| \leq R_g$. Finally we define the *level margin* of a weight vector $\mathbf{w}$ for a training set as

$$\gamma = \min_{\{(v,v')\}} \mathbf{w} \cdot \Big(\phi(v) - \phi(v')\Big). \tag{5.15}$$

Here, the set $\{(v, v')\}$ contains any pair such that v is y-good, v' is not y-good, and v and v' are at the same search level. The level margin denotes the minimum score gap between a y-good and a y-bad node at the same search level.

The priority function used to rank the search nodes is defined as $p_{\mathbf{w}}(v) = g(v) - \mathbf{w} \cdot \phi(v)$. Smaller priority function value ranks higher during search. With these definitions, we have the following theorem:

Theorem 5.1 (Speedup mistake bound)**.** *Given a training set such that there exists a weight vector* $\mathbf{w}$ *with level margin* $\gamma > 0$ *and* $\|\mathbf{w}\| = 1$*, the speedup learning algorithm (Algorithm 1) will converge with a consistent weight vector after making no more than* $\frac{R_\phi^2 + 2R_g}{\gamma^2}$ *weight updates.*

5.4.2 Proof of mistake bound theorem

The goal of this section is to prove Theorem 5.1 in the previous section. We will prove two lemmas, which will lead to the final proof of the theorem. Before introducing the two lemmas, we repeat the relevant definitions here for convenience.

Let R_ϕ be a constant such that for every pair of search nodes (v, v'), $\|\phi(v) - \phi(v')\| \leq R_\phi$. Let R_g be a constant such that for every pair of search nodes (v, v'), $|g(v) - g(v')| \leq R_g$. Finally, we define the *level margin* of a weight vector $\mathbf{w}$ for a training set as

$$\gamma = \min_{\{(v,v')\}} \mathbf{w} \cdot \Big(\phi(v) - \phi(v')\Big). \tag{5.16}$$

Here, the set $\{(v, v')\}$ contains any pair such that v is y-good, v' is not y-good, and v and v' are at the same search level. The priority functin used to rank the search nodes is defined

as $p_{\mathbf{w}}(v) = g(v) - \mathbf{w} \cdot \phi(v)$. Smaller priority function value ranks higher during search. With these definitions we have the following two lemmas:

Lemma 5.1. *Let $\mathbf{w}^k$ be the weights before the k^{th} mistake is made ($\mathbf{w}^1 = \mathbf{0}$). Right after the k^{th} mistake is made, the norm of the weight vector $\mathbf{w}^{k+1}$ has the following upper bound:*

$$\|\mathbf{w}^{k+1}\|^2 \leq \|\mathbf{w}^k\|^2 + R_\phi^2 + 2R_g$$

Proof. When the k^{th} mistake is made, we get $\mathbf{w}^{k+1}$ by using the update rule in either line 11 or line 14 of the algorithm. Let us consider the case of updating using line 11 first. We have

$$\begin{aligned}
\|\mathbf{w}^{k+1}\|^2 &= \|\mathbf{w}^k + \phi(v^*) - \frac{1}{|B|}\sum_{v\in B}\phi(v)\|^2 \\
&= \|\mathbf{w}^k\|^2 + \|\phi(v^*) - \frac{1}{|B|}\sum_{v\in B}\phi(v)\|^2 \\
&\quad + 2\mathbf{w}^k \cdot \Big(\phi(v^*) - \frac{1}{|B|}\sum_{v\in B}\phi(v)\Big). \qquad (5.17)
\end{aligned}$$

We will upper bound each term separately on the right side of Eq. (5.17). To bound the second term, we use the definition of R_ϕ and the properties of vector dot product:

$$\begin{aligned}
&\|\phi(v^*) - \frac{1}{|B|}\sum_{v\in B}\phi(v)\|^2 \\
&= \frac{1}{|B|^2}\|\sum_{v\in B}(\phi(v^*) - \phi(v))\|^2 \\
&= \frac{1}{|B|^2}\sum_{v\in B}(\phi(v^*) - \phi(v)) \cdot \sum_{v'\in B}(\phi(v^*) - \phi(v')) \\
&= \frac{1}{|B|^2}\sum_{v,v'\in B}(\phi(v^*) - \phi(v)) \cdot (\phi(v^*) - \phi(v')) \\
&\leq \frac{1}{|B|^2}\sum_{v,v'\in B}\|\phi(v^*) - \phi(v)\|\|\phi(v^*) - \phi(v')\| \\
&\leq \frac{1}{|B|^2}\sum_{v,v'\in B}R_\phi^2 \\
&= R_\phi^2. \qquad (5.18)
\end{aligned}$$

To bound the third term on the right side of Eq. (5.17), note that the update is happening because of the k^{th} mistake is being made. Therefore for any node $v \in B$, we have (smaller priority function value ranks higher)

$$p_{\mathbf{w}^k}(v^*) \geq p_{\mathbf{w}^k}(v).$$

Using the definitions of $p_{\mathbf{w}^k}(v)$ and R_g,

$$\mathbf{w}^k \cdot \Big(\phi(v^*) - \phi(v)\Big) \leq g(v^*) - g(v) \leq R_g.$$

Now we have the upper bound for the third term on the right side of Eq. (5.17):

$$\begin{aligned}
&2\mathbf{w}^k \cdot \Big(\phi(v^*) - \frac{1}{|B|}\sum_{v\in B}\phi(v)\Big) \\
&= \frac{2}{|B|}\mathbf{w}^k \cdot \sum_{v\in B}\Big(\phi(v^*) - \phi(v)\Big) \\
&= \frac{2}{|B|}\sum_{v\in B}\mathbf{w}^k \cdot \Big(\phi(v^*) - \phi(v)\Big) \\
&\leq \frac{2}{|B|}\sum_{v\in B} R_g \\
&= 2R_g. \qquad (5.19)
\end{aligned}$$

Combining Eqs. (5.17), (5.18), and (5.19) leads to the following:

$$\|\mathbf{w}^{k+1}\|^2 \leq \|\mathbf{w}^k\|^2 + R_\phi^2 + 2R_g.$$

Next we consider the case of updating using line 14 of the algorithm, in which we have

$$\begin{aligned}
\|\mathbf{w}^{k+1}\|^2 &= \|\mathbf{w}^k + \phi(v^*) - \phi(\hat{v})\|^2 \\
&= \|\mathbf{w}^k\|^2 + \|\phi(v^*) - \phi(\hat{v})\|^2 \\
&\quad + 2\mathbf{w}^k \cdot \Big(\phi(v^*) - \phi(\hat{v})\Big). \qquad (5.20)
\end{aligned}$$

Using the definition of R_ϕ,

$$\|\phi(v^*) - \phi(\hat{v})\|^2 \leq R_\phi^2. \qquad (5.21)$$

Also, since a mistake is made by the weight $\mathbf{w}^k$, we know $p_{\mathbf{w}^k}(v^*) \geq p_{\mathbf{w}^k}(\hat{v})$, which implies

$$\mathbf{w}^k \cdot \Big(\phi(v^*) - \phi(\hat{v})\Big) \leq g(v^*) - g(\hat{v}) \leq R_g. \qquad (5.22)$$

Combining Eqs (5.20), (5.21), and (5.22) again leads to

$$\|\mathbf{w}^{k+1}\|^2 \leq \|\mathbf{w}^k\|^2 + R_\phi^2 + 2R_g.$$

□

Lemma 5.2. *Let $\mathbf{w}^k$ be the weights before the k^{th} mistake is made ($\mathbf{w}^1 = \mathbf{0}$). Let $\mathbf{w}$ be a weight vector with level margin γ as defined in Eq.* (5.16)*. Then*

$$\mathbf{w} \cdot \mathbf{w}^{k+1} \geq \mathbf{w} \cdot \mathbf{w}^k + \gamma.$$

Proof. First consider the update rule of line 11 of the algorithm,

$$\begin{aligned}
\mathbf{w} \cdot \mathbf{w}^{k+1} &= \mathbf{w} \cdot \Big(\mathbf{w}^k + \phi(v^*) - \frac{1}{|B|} \sum_{v \in B} \phi(v) \Big) \\
&= \mathbf{w} \cdot \mathbf{w}^k + \frac{1}{|B|} \sum_{v \in B} \mathbf{w} \cdot \Big(\phi(v^*) - \phi(v) \Big) \\
&\geq \mathbf{w} \cdot \mathbf{w}^k + \frac{1}{|B|} \sum_{v \in B} \gamma \\
&= \mathbf{w} \cdot \mathbf{w}^k + \gamma,
\end{aligned}$$

where we use the definition of the level margin γ. Next consider the update rule of line 14 of the algorithm,

$$\begin{aligned}
\mathbf{w} \cdot \mathbf{w}^{k+1} &= \mathbf{w} \cdot \Big(\mathbf{w}^k + \phi(v^*) - \phi(\hat{v}) \Big) \\
&= \mathbf{w} \cdot \mathbf{w}^k + \mathbf{w} \cdot \Big(\phi(v^*) - \phi(\hat{v}) \Big) \\
&\geq \mathbf{w} \cdot \mathbf{w}^k + \gamma.
\end{aligned}$$

□

Now we are ready to prove Theorem 5.1 of the main paper, which is repeated here for convenience.

Theorem 5.2 (Speedup mistake bound)**.** *Given a training set such that there exists a weight vector* $\mathbf{w}$ *with level margin* $\gamma > 0$ *and* $\|\mathbf{w}\| = 1$*, the speedup learning algorithm (Algorithm 1) will converge with a consistent weight vector after making no more than* $\frac{R_\phi^2 + 2R_g}{\gamma^2}$ *weight updates.*

Proof. Let $\mathbf{w}^k$ be the weights before the k^{th} mistake is made ($\mathbf{w}^1 = \mathbf{0}$). Using Lemma 5.1 repetitively (induction on k) gives us

$$\|\mathbf{w}^{k+1}\|^2 \leq k(R_\phi^2 + 2R_g).$$

Similarly, induction on k using Lemma 5.2,

$$\mathbf{w} \cdot \mathbf{w}^{k+1} \geq k\gamma.$$

Finally we have

$$1 \geq \frac{\mathbf{w} \cdot \mathbf{w}^{k+1}}{\|\mathbf{w}\| \|\mathbf{w}^{k+1}\|} \geq \frac{k\gamma}{\sqrt{k(R_\phi^2 + 2R_g)}},$$

which gives us

$$k \leq \frac{R_\phi^2 + 2R_g}{\gamma^2}.$$

□

5.4.3 Avoiding computing the input features

So far, we have shown that a structured prediction problem can be converted to a beam search problem. The priority function for ranking search nodes is determined by $p(v) = g(v) + h(v)$. We have seen how the h function be trained to enforce structural constraints. However, there are other opportunities for speeding up as well.

Computing the path cost $g(v)$ involves calculating the corresponding ILP coefficients, which in turn requires feature extraction using the original trained model. This is usually a time-consuming step [97], and thus motivating the question of whether we can avoid calculating them without losing accuracy. If a search node is strongly preferred by the heuristic function, the path cost is unlikely to reverse the heuristic function's decision. In this case, we can rank the candidate search nodes with heuristic function only.

Formally, given a fixed beam size b and the beam candidates C_t at step t from which we need to select the beam B_t, we can rank the nodes in C_t from smallest to largest according to the heuristic function value $h(v)$. Denote the b^{th} smallest node as v_b and the $(b+1)^{\text{th}}$ smallest node as v_{b+1}, we define the heuristic gap Δ_t as

$$\Delta_t = h(v_{b+1}) - h(v_b). \tag{5.23}$$

If the beam B_t is selected from C_t only according to heuristic function, then Δ_t is the gap between the last node in the beam and the first node outside the beam. Next we define the path-cost gap δ_t as

$$\delta_t = \max_{v, v' \in C_t} (v - v'). \tag{5.24}$$

With these definitions we immediately have the following theorem:

Theorem 5.3. *Given the beam candidates C_t with heuristic gap Δ_t and path-cost gap δ_t, if $\Delta_t > \delta_t$, then using only heuristic function to select the beam B_t will have the same set of nodes selected as using the full priority function up to their ordering in the beam.*

Proof. Let $v^* \in C_t$ be any node that is ranked within top-b nodes by the heuristic function $h(\cdot)$, and let $v \in C_t$ be an arbitrary node that is *not* within top-b nodes. By definitions of v_b and v_{b+1} we have

$$h(v^*) \leq h(v_b) \leq h(v_{b+1}) \leq h(v).$$

Therefore,

$$h(v) - h(v^*) \geq h(v_{b+1}) - h(v_b) = \Delta_t. \tag{5.25}$$

Also by definition of δ_t we have

$$g(v) - g(v^*) \geq -\delta_t. \tag{5.26}$$

Combining Eqs. (5.25) and (5.26),

$$\begin{aligned} p(v) - p(v^*) &= \Big(g(v) + h(v)\Big) - \Big(g(v^*) + h(v^*)\Big) \\ &= \Big(h(v) - h(v^*)\Big) + \Big(g(v) - g(v^*)\Big) \\ &\geq \Delta_t - \delta_t \\ &> 0. \end{aligned}$$

Thus, for any node $v^* \in C_t$, if it is selected in the beam B_t (ranked top b) by the heuristic funciton $h(\cdot)$, it will be selected in the beam B_t by the full priority funciton $p(\cdot)$. □

If the condition of Theorem 5.3 holds, then we can rank the candidates using only heuristic function without calculating the path cost. This will further save computation time. However, without actually calculating the path cost there is no way to determine the path-cost gap δ_t at each step. In practice we can treat δ_t as an empirical parameter θ and define the following priority function

$$p_\theta(v) = \begin{cases} h(v), & \text{if } \Delta_t > \theta, \\ g(v) + h(v), & \text{otherwise.} \end{cases} \tag{5.27}$$

5.5 Experiments

We empirically evaluate the speedup based inference scheme described in Section 5.4 on the problem of predicting entities and relations (i.e., our running example). In this task, we are asked to label each entity and the relationship between each pair of the entities. We assume the entity candidates are given, either from human annotators or from

a preprocessing step. The goal of inference is to determine the types of the entity spans, and the relations between them, as opposed to identify entity candidates. The research questions we seek to resolve empirically are the following:

1. Does using a learned speedup heuristic recover structurally valid outputs without paying the inference cost of the integer linear program solver?

2. Can we construct accurate outputs without always computing the input features and using only the learned heuristic to guide search?

The dataset we used is from the previous work by [85]. It contains 1441 sentences. Each sentence contains several entities with labels and the labeled relations between every pair of entity. There are three types of entities, `person`, `location`, and `organization`, and five types of relations, `Kill`, `LiveIn`, `WorkFor`, `LocatedAt`, and `OrgBasedIn`. There are two constraints associated with each relation type, specifying the allowed source and target arguments. For example, if the relation label is `LiveIn`, the source entity must be `person` and the target entity must be `location`. There is also another kind of constraint that says for every pair of entities, they can not have a relation label in both directions between them: one of the direction must be labeled as `NoRel`.

We reimplemented the model from the original work using the same set of features as for the entity and relation scoring functions. We used 70% of the labeled data to train an ILP-based inference scheme, which will become our black-box solver for learning the speedup classifier. The remaining 30% labeled data are held out for evaluations.

We use 29950 sentences from the Gigaword corpus [34] to train the speedup classifier. The entity candidates are extracted using the Stanford Named Entity Recognizer [59]. We ignore the entity labels, however, since our task requires determining the type of the entities and relations. The features we use for the speedup classifiers are counts of the pairs of labels of the form (source label, relation label), (relation label, target label), and counts of the triples of labels of the form (source label, relation label, target label). We run Algorithm 1 over this unlabeled dataset, and evaluate the resulting speedup classifier on the held out test set. In all of our speedup search implementations, we first assign labels to the entities from left to right, then the relations among them.

We evaluate the learned speedup classifier in terms of both accuracy and speed. The accuracy of the speedup classifier can be evaluated using three kinds of metrics: F-1 scores against gold labels, F-1 scores against the ILP solver's prediction, and the *validity ratio*, which is the percentage of the predicted examples agreeing with all constraints. All our experiments were conducted on a server with eight Intel i7 3.40 GHz cores and 16G memory. We disabled multi-threaded execution in all cases for a fair comparison.

5.5.1 Evaluation of Algorithm 1

Our first set of experiments evaluates the impact of Algorithm 1. These results are shown in Table 5.1. We see the ILP solver achieves perfect entity and relation F-1 when compared with the ILP model itself. It guarantees that all constraints are satisfied. Its accuracy against the gold label and its prediction time become the baselines of our speedup classifiers. We also provide two search baselines. The first search baseline uses a greedy search without any constraint considerations. In this setting, each label is assigned independently since the step cost of assigning a label to an entity or a relation variable depends only on the corresponding coefficients in the ILP objectives. In this case, a structured prediction problem becomes several independent multi-class classification problems. The prediction time is faster than ILP, but the validity ratio is rather low (0.29). The second search baseline is a greedy search with constraint satisfaction. The constraints are guaranteed to be satisfied by using the standard arc-consistency search. The prediction takes much longer than the ILP solver (844 ms vs. 239 ms.).

We trained a speedup classifier with two different beam sizes. Even with beam width $b = 1$, we can obtain $> 95\%$ validity ratio, and the prediction time is much faster than the

Table 5.1. Performance of the speedup classifier with different beam sizes, compared with the ILP solver and search without heuristics. CPU time is in milli-seconds, and averaged over five different runs with standard deviations.

Model	Ent. F1 Gold	Rel. F1 Gold	Ent. F1 ILP	Rel. F1 ILP	Time (ms)	Valid Ratio
ILP	0.827	0.482	1.000	1.000	239 ± 11.4	1.00
search	0.822	0.334	0.877	0.546	170 ± 6.5	0.29
constrained search	0.800	0.572	0.880	0.741	844 ± 31.0	1.00
speedup ($b = 1$)	0.822	0.447	0.877	0.674	136 ± 2.1	0.96
speedup ($b = 2$)	0.844	0.484	0.930	0.752	158 ± 19.8	0.95

ILP model. Furthermore, we see that the F-1 score evaluated against gold labels is only slightly worse than the ILP model. With beam width $b = 2$, we recover the ILP model accuracy when evaluated against gold labels. The prediction time is still much less than the ILP solver.

5.5.2 Experiments on ignoring the model cost

In this section, we empirically verify the idea that we do not always need to compute the path cost if the heuristic gap Δ_t is large. We use the evaluation function $p_\theta(v)$ in Eq. (5.27) with different values of θ to rank the search nodes. The results are given in Table 5.2.

For both beam widths, $\theta = 0$ is the case in which the original model is completely ignored. All the nodes are ranked using the speedup heuristic function only. Even though it has perfect validity ratio, the result is rather poor when evaluated on F-1 scores. When θ increases, the entity and relation F-1 scores quickly jump up, essentially getting back the same accuracy as the speedup classifiers in Table 5.1. However, the prediction time is lowered compared to the results from Table 5.1.

5.6 Discussion and related work

The idea of learning memo functions to make computation more efficient goes back to [63]. Speedup learning has been studied since the 1980s in the context of general problem solving, where the goal is to learn a problem solver that becomes faster as opposed to becoming more accurate as it sees more data. Alan Fern [25] gives a broad survey of this area. In this paper, we presented a variant of this idea that is more concretely applied to

Table 5.2. Performance of the speedup classifier with different beam size and θ values. CPU Time is in milli-second, and averaged over five different runs with standard deviations.

Beam size	θ	Ent. F1 Gold	Rel. F1 Gold	Ent. F1 ILP	Rel. F1 ILP	Time (ms)	Valid Ratio
$b = 1$	0	0.21	0	0.173	0	39 ± 3.7	1.00
$b = 1$	0.25	0.822	0.435	0.877	0.546	87 ± 2.5	0.99
$b = 1$	0.5	0.822	0.455	0.877	0.672	114 ± 4.7	0.98
$b = 2$	0	0.373	0.152	0.377	0.139	55 ± 2.5	1.00
$b = 2$	0.25	0.819	0.461	0.893	0.623	130 ± 14.4	0.99
$b = 2$	0.5	0.825	0.494	0.907	0.689	134 ± 4.0	0.98

structured output prediction.

Efficient inference is a central topic in structured prediction. In order to achieve efficiency, various strategies are adopted in the literature. Search based strategies are commonly used for this purpose and several variants abound. The idea of framing a structured prediction problem as a search problem has been explored by several previous works [14, 20, 21, 23, 42]. It usually admits incorporating arbitrary features more easily than fully global structured prediction models like conditional random fields [49], structured perceptron [13], and structured support vector machines [101, 103]. In such cases too, the inference can be solved approximately using heuristic search. Either a fixed beam size [110], or a dynamically-sized beam [5] can be used. In our work, we fix the beam size. The key difference from previous work is that our ranking function combines information from the trained model with the heuristic function, which characterizes constraint information. Closely related to the work described in this paper are approaches that learn to prune the search space [38, 105] and learn to select features [37].

Another line of recent related work focuses on discovering the problem-level regularities across the inference space. These *amortized* inference schemes are designed using deterministic rules for discovering when a new inference problem can reuse previously computed solutions [48, 98] or in the context of a Bayesian network by learning a stochastic inverse network that generates outputs [99].

Our work is also related to the idea of imitation learning [7, 20, 83, 84]. In this setting, we are given a reference policy, which may or may not be a good policy. The goal of learning is to learn another policy to imitate the given policy or even learn a better one. Learning usually proceeds in an online fashion. However, imitation learning requires learning a new policy, which is independent of the given reference policy, since, during test time, the reference policy is no longer available. In our case, we can think of the black-box solver as a reference policy. During prediction, we always have this solver at our disposal; what we want is avoiding unnecessary calls to the solver. Following recent successes in imitation learning, we expect that we can replace the linear heuristic function with a deep network to avoid feature design.

Also related is the idea of knowledge distillation [6, 40, 43] that seeks to train a student classifier (usually a neural network) to compress and mimic a larger teacher network, thus

improve prediction speed. The primary difference with the speedup idea of this paper is that our goal is to be more efficient at constructing internally self-consistent structures without explicitly searching over the combinatorially large output space with complex constraints.

5.7 Conclusions

In this paper, we asked whether we can learn to make inference faster over the lifetime of a structured output classifier. To address this question, we developed a search-based strategy that learns to mimic a black-box inference engine but is substantially faster. We further extended this strategy by identifying cases where the learned search algorithm can avoid expensive input feature extraction to further improve speed without losing accuracy. We empirically evaluated our proposed algorithms on the problem of extracting entities and relations from text. Despite using an object-heavy JVM-based implementation of search, we showed that by exploiting regularities across the output space, we could outperform the industrial strength Gurobi integer linear program solver in terms of speed, while matching its accuracy.

CHAPTER 6

DECISION BOUNDARIES OF DEEP NEURAL NETWORKS WITH PIECEWISE LINEAR ACTIVATIONS

Chapter 3 discusses the expressiveness of shallow rectifier networks. The decision boundary of a shallow rectifier network is given in terms of an equivalent condition specifying when an example will be labeled as positive or negative. In this chapter, we extend that results from two perspectives. First, we consider *deep* rectifier networks; then we consider more general piecewise linear activation functions. We identify all possible hyperplanes making up the decision boundaries and show that the number of hyperplanes can be exponential in the number of hidden units of the network. We further provide an approach to construct the decision boundaries using these hyperplanes and obtain a Boolean function, which is equivalent to the original neural network. The proposed approach can be applied to a family of networks with piecewise linear activation functions such as ReLU, leaky ReLU, and maxout, etc.

6.1 Feedforward rectifier networks

In this section, we formally define what is a feedforward rectifier network, and discuss the relation between its linear regions, when used as a regression function, and its decision boundary, when used as a binary classification function.

6.1.1 Definitions

We begin by defining the *feedforward rectifier network*.

Definition 6.1. *A feedforward rectifier network with L hidden layers and one output unit is a function* $F(\mathbf{x};\theta) : \mathbb{R}^{n_0} \to \mathbb{R}$ *in the following form:*

$$\begin{aligned}
\mathbf{h}^{(0)} &= \mathbf{x} \\
\mathbf{z}^{(l)} &= \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \quad l \in [L] \\
\mathbf{h}^{(l)} &= \text{rect}(\mathbf{z}^{(l)}), \quad l \in [L] \\
y &= \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + b^{(L+1)}
\end{aligned} \tag{6.1}$$

where $\mathbf{x} \in \mathbb{R}^{n_0}$ *is the input to the network, and* $y \in \mathbb{R}$ *is the output.* $\mathbf{z}^{(l)}$ *is the preactivation value of the* l^{th} *hidden layer, and* $\mathbf{h}^{(l)}$ *is the corresponding activated value. Assume this network has* n_0 *input units,* n_l *units in the* l^{th} *hidden layer, and one output unit. The parameters* θ *of the network are* $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$, $\mathbf{W}^{(L+1)} \in \mathbb{R}^{1 \times n_L}$, *and* $b^{(L+1)} \in \mathbb{R}$.

The function defined in Definition 6.1 is a regression function; that is, the output y is a real number. However, in this chapter, we are mostly interested in using the network as a classification function. In this case, we implicitly assume that the output value y is thresholded at zero. The input example $\mathbf{x}$ is classified as positive if the corresponding output $y \geq 0$, and negative if $y < 0$. Therefore, the decision boundary of this network is specified by the equation $F(\mathbf{x}; \theta) = 0$.

6.1.2 Linear regions and decision boundaries

The rectifier network in Definition 6.1 is a piecewise linear function, since every operation in Eq. (6.1) is linear except the rectifier activation, $\text{rect}(\cdot)$, which is piecewise linear. One way to capture the complexity of a piecewise linear function is to count the number of linear regions of the given function, as shown by [64, 73, 91]. Since we are interested in using the network as classification functions, we can directly study the decision boundary of the function computed by the rectifier network.

Definition 6.2. *The decision boundary of a function computed by the rectifier network is the set of input points* $\mathbf{x}$ *whose corresponding output value* $F(\mathbf{x}; \theta) = 0$,

$$\{\mathbf{x} \in \mathbb{R}^{n_0} : F(\mathbf{x}; \theta) = 0\}$$

To obtain the decision boundary of the network, we can set the function value $F(\mathbf{x}; \theta) = 0$ in every linear regions. Denote the linear regions of the function computed by the network as $\{R_1, \ldots, R_t\}$, where t is the number of linear regions. Within each region R_i, the function $F(\mathbf{x}; \theta)$ is just a linear function $F_i(\mathbf{x}; \theta) = \mathbf{w}_i \cdot \mathbf{x} + b_i$. Except the case in

which $F_i(\mathbf{x};\theta)$ is always positive or negative in R_i, the hyperplane equation $\mathbf{w}_i \cdot \mathbf{x} + b_i = 0$ specifies one piece of the decision boundary of the given rectifier network in region R_i. We call the hyperplane $\mathbf{w}_i \cdot \mathbf{x} + b_i = 0$ a *decision hyperplane*. The overall decision boundary of the network is composed of all such decision hyperplanes. We immediately have the following fact.

Proposition 6.1. *The function computed by a feedforward rectifier network is a piecewise linear function with t linear regions. The number of decision hyperplanes is at most t.*

Proof. If in region R_i the function $F_i(\mathbf{x};\theta) = \mathbf{w}_i \cdot \mathbf{x} + b_i$ is always positive or negative, there is no boundary in this region. Otherwise, there is exactly one decision hyperplane $\mathbf{w}_i \cdot \mathbf{x} + b_i = 0$, which is part of the overall decision boundary. Therefore, the number of decision hyperplanes is at most t. □

By virtue of Proposition 6.1, any upper bound on the number of linear regions is also an upper bound on the number of decision boundary hyperplanes. To our best knowledge, the tightest upper bound on the number of linear regions is given by [91], which leads to the following corollary,

Corollary 6.1. *The number of decision hyperplanes in the rectifier network in Definition 6.1 is at most*

$$\sum_{(j_1,\ldots,j_L)\in J} \prod_{l=1}^{L} \binom{n_l}{j_l}$$

where $J = \{(j_1,\ldots,j_L) \in \mathbb{Z}^L : 0 \leq j_l \leq \min\{n_0, n_1 - j_1, \ldots, n_{l-1} - j_{l-1}, n_l\}, \forall l \in [L]\}$.

Example 6.1. *Consider the following one-layer rectifier network with input dimension $n_0 = 2$*

$$y = 1 - \text{rect}(x_1) - \text{rect}(x_2) + \text{rect}(x_1 + x_2)$$

As shown in Figure 6.1, it has six linear regions, delimit by three hyperplanes $x_1 = 0$, $x_2 = 0$, and $x_1 + x_2 = 0$. Only four linear regions contain decision hyperplanes. The hyperplane equation of the decision boundary can be obtained by setting the value of the linear function to be zero in each region. For example, in the region where the function is $y = 1 + x_1$, the decision boundary in this region is $1 + x_1 = 0$.

From Example 6.1, we see that the number of decision hyperplanes is less than the number of linear regions. We also see that the decision boundary can be disconnected;

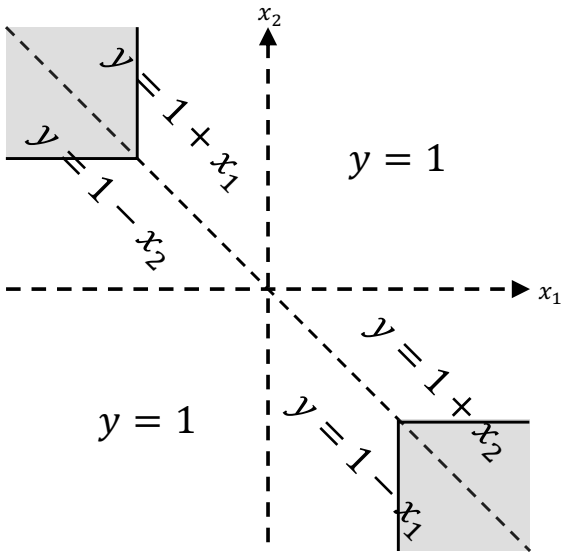

Figure 6.1. Illustration of Example 6.1. The network is given by $y = 1 - \text{rect}(x_1) - \text{rect}(x_2) + \text{rect}(x_1 + x_2)$. Dashed lines delimit linear regions, and solid lines specify decision boundary. Within each linear region, the corresponding linear function is also displayed. Inputs from shaded area are classified as negative.

there are two disconnected negative regions in this example, indicated by shaded areas in Figure 6.1.

It is worth pointing out that usually we do not know the arrangement of linear regions of a given rectifier network; therefore, we do not know how different pieces of decision hyperplanes $\mathbf{w}_i \cdot \mathbf{x} + b_i = 0$ are combined to form the overall decision boundary. The goal of this chapter is not only to identify possible decision hyperplanes but also to analyze how those hyperplanes are combined to form the decision boundary.

6.2 Decision boundaries of feedforward rectifier networks

In this section, we introduce our approach of constructing the decision boundaries of feedforward rectifier networks. We first review the decision boundary of a shallow rectifier network in which there is only one hidden layer, which we introduced in Chapter 3. We show under what condition a given input example is classified as positive. For a deep rectifier network, we recursively apply the shallow-network result to obtain its decision boundary.

6.2.1 Shallow rectifier networks

In this section, we use shallow rectifier network to refer to the feedforward rectifier network with only one hidden layer. The result of this section is essentially the same as the one in Section 3.1.2. Here we restate the relevant definitions and theorems in a notation that facilitates an easy transition into the case of deep rectifier networks. We begin with the following definitions.

Definition 6.3. *A shallow rectifier network is a function $F(\mathbf{x};\theta) : \mathbb{R}^{n_0} \to \mathbb{R}$ in the following form:*

$$\begin{aligned} \mathbf{z} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \mathbf{h} &= \text{rect}(\mathbf{z}) \\ y &= \mathbf{W}^{(2)}\mathbf{h} + b^{(2)}, \end{aligned} \tag{6.2}$$

where $\mathbf{x} \in \mathbb{R}^{n_0}$ is the input to the network, and $y \in \mathbb{R}$ is the output. $\mathbf{z}$ is the preactivation value of the hidden layer, and $\mathbf{h}$ is the corresponding activated value. Assume this network has n_1 hidden units. The parameters θ of the network are $\mathbf{W}^{(1)} \in \mathbb{R}^{n_1 \times n_0}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{n_1}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{1 \times n_1}$, and $b^{(2)} \in \mathbb{R}$.

Definition 6.4. *Given a shallow rectifier network in Definition 6.3, we define the set of positive and negative hidden units as*

$$\begin{aligned} \mathcal{U}_+ &= \{i_1 \in [n_1] : W^{(2)}_{1,i_1} > 0\}, \\ \mathcal{U}_- &= \{i_1 \in [n_1] : W^{(2)}_{1,i_1} < 0\}. \end{aligned}$$

$\mathcal{U}_+$ contains the indices of the hidden units whose weights are positive, and $\mathcal{U}_-$ contains those with negative weights.

Definition 6.5. *Consider a shallow rectifier network in Definition 6.3. For a subset $\mathcal{S} \subseteq [n_1]$, define a boolean function $B(\mathbf{x};\mathcal{S})$. $B(\mathbf{x};\mathcal{S})$ is true if $\mathbf{x}$ satisfies the following inequality:*

$$\sum_{i_1 \in \mathcal{S}} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{(1)}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{S}} W^{(2)}_{1,i_1} b^{(1)}_{i_1} + b^{(2)} \geq 0, \tag{6.3}$$

or equivalently,

$$\sum_{i_1 \in \mathcal{S}} W^{(2)}_{1,i_1} z_{i_1} + b^{(2)} \geq 0, \tag{6.4}$$

where $\mathbf{z}$ is related to $\mathbf{x}$ through Eq. (6.21).

Note that the inequality in (6.3) is linear on $\mathbf{x}$, therefore it specifies a hyperplane in the n_0-dimensional input space. $B(\mathbf{x}; \mathcal{S})$ is true if $\mathbf{x}$ is at one side of the hyperplane and false if $\mathbf{x}$ is at the other side. We will see that the hyperplanes specified by $B(\mathbf{x}; \mathcal{S})$ for all nonempty subset $\mathcal{S} \subseteq [n_1]$ are candidates for decision hyperplanes.

Now we are ready to state the decision boundary of the shallow rectifier network in the following theorem:

Theorem 6.1. *Consider the shallow rectifier network given in Definition 6.3. For any input* $\mathbf{x}$*, the corresponding output* $y \geq 0$ *if and only if there exists a subset* $\mathcal{S}_+ \subseteq \mathcal{U}_+$ *such that for all* $\mathcal{S}_- \subseteq \mathcal{U}_-$*, the value of* $B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-)$ *is true. We can write this equivalence condition in the following logic form:*

$$y \geq 0 \iff \bigvee_{\mathcal{S}_+ \subseteq \mathcal{U}_+} \bigwedge_{\mathcal{S}_- \subseteq \mathcal{U}_-} B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-), \tag{6.5}$$

where the disjunction is over all possible subsets $\mathcal{S}_+ \subseteq \mathcal{U}_+$*, and the conjunction is over all possible subsets* $\mathcal{S}_- \subseteq \mathcal{U}_-$*.*

Theorem 6.1 is a restatement of the Theorem 3.1, and the proof is skipped here. In Chapter 3 we provide an example in Eq. 3.4 to illustrate the application of the theorem. Here we provide another example.

Example 6.2. *Consider the same network as given in Example 6.1.*

$$y = 1 - \text{rect}(x_1) - \text{rect}(x_2) + \text{rect}(x_1 + x_2)$$

We index the rectifier units in the order of $\text{rect}(x_1)$*,* $\text{rect}(x_2)$*, and* $\text{rect}(x_1 + x_2)$*. According to Definition 6.4, the first two rectifier units have negative weights and the third rectifier unit has a positive weight. Hence* $\mathcal{U}_+ = \{3\}$ *and* $\mathcal{U}_- = \{1, 2\}$*.* $\mathcal{U}_+$ *has two subsets,* $\emptyset$ *and* $\{3\}$*; and* $\mathcal{U}_-$ *has four subsets,* $\emptyset$*,* $\{1\}$*,* $\{2\}$ *and* $\{1, 2\}$*. Theorem 6.1 says an input* $\mathbf{x} = (x_1, x_2)$ *is classified as positive if and only if at least one system of inequalities holds,*

$$\begin{cases} 1 \geq 0, & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \emptyset \text{ and } \mathcal{S}_- = \emptyset \\ 1 - x_1 \geq 0, & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \emptyset \text{ and } \mathcal{S}_- = \{1\} \\ 1 - x_2 \geq 0, & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \emptyset \text{ and } \mathcal{S}_- = \{2\} \\ 1 - x_1 - x_2 \geq 0. & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \emptyset \text{ and } \mathcal{S}_- = \{1, 2\} \end{cases} \tag{6.6}$$

or

$$\begin{cases} 1 + x_1 + x_2 \geq 0, & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \{3\} \text{ and } \mathcal{S}_- = \emptyset \\ 1 + x_2 \geq 0, & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \{3\} \text{ and } \mathcal{S}_- = \{1\} \\ 1 + x_1 \geq 0, & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \{3\} \text{ and } \mathcal{S}_- = \{2\} \\ 1 \geq 0. & B(\mathbf{x}; \mathcal{S}_+ \cup \mathcal{S}_-) \text{ with } \mathcal{S}_+ = \{3\} \text{ and } \mathcal{S}_- = \{1, 2\}. \end{cases} \tag{6.7}$$

It is easy to verify that the conditions given in inequalities (6.6) *or* (6.7) *agree with the unshaded region in Figure 6.1. Also observe that there are six nontrivial inequalities in* (6.6) *and* (6.7)*, whereas the final decision boundary in Figure 6.1 is composed from only four hyperplanes:* $1 \pm x_1 = 0$ *and* $1 \pm x_2 = 0$*. This means that even though the conditions given in inequalities* (6.6) *or* (6.7) *correctly specify the decision boundary, it may not be the simplest way of doing so. This is often the case when applying Theorem 6.1 to a shallow rectifier network.*

If the sizes of the positive and negative subsets are m_1 and m_2, respectively (i.e, $m_1 = |\mathcal{U}_+|$ and $m_2 = |\mathcal{U}_-|$), then $\mathcal{U}_+$ has 2^{m_1} subsets and $\mathcal{U}_-$ has 2^{m_2} subsets. Thus, there are $2^{m_1+m_2}$, or 2^{n_1} such Boolean functions.[1] The expression in Eq. (6.5) is in the disjunctive normal form (DNF), where each conjunct contains 2^{m_2} B's and there are 2^{m_1} such terms. Since each B specify a hyperplane in the input space, this characterizes the decision boundary of the shallow rectifier network as a DNF expression over these hyperplane decisions. As we show in Example 6.2, the actual decision boundary may not need all 2^{n_1} hyperplanes.

A special case of a shallow rectifier network is when all output weights are positive ($\forall i_1, W^{(2)}_{1,i_1} > 0$), or all output weights are negative ($\forall i_1, W^{(2)}_{1,i_1} < 0$). Applying Theorem 6.1 to such networks leads to the following corollary.

Corollary 6.2. *Consider the shallow rectifier network given in Definition 6.3. Suppose for all* $i_1 \in [n_1]$, $W^{(2)}_{1,i_1} > 0$*. For any input* $\mathbf{x}$*, the corresponding output* $y \geq 0$ *if and only if there exists a subset* $\mathcal{S} \subseteq [n_1]$ *such that the value of* $B(\mathbf{x}; \mathcal{S})$ *is true. We can write this equivalence condition in the following logic form:*

$$y \geq 0 \iff \bigvee_{\mathcal{S} \subseteq [n_1]} B(\mathbf{x}; \mathcal{S}),$$

where the disjunction is over all possible subsets $\mathcal{S} \subseteq [n_1]$*.*

Similarly, if for all $i_1 \in [n_1]$, $W^{(2)}_{1,i_1} < 0$*, then for any input* $\mathbf{x}$*, the corresponding output* $y \geq 0$ *if and only if for all subsets* $\mathcal{S} \subseteq [n_1]$ *the value of* $B(\mathbf{x}; \mathcal{S})$ *is true. We can write this equivalence condition in the following logic form:*

[1] Assume for all i_1, $W^{(2)}_{1,i_1} \neq 0$, otherwise the unit i_1 can be deleted. Hence $m_1 + m_2 = n_1$.

$$y \geq 0 \iff \bigwedge_{\mathcal{S} \subseteq [n_1]} B(\mathbf{x}; \mathcal{S}),$$

where the conjunction is over all possible subsets $\mathcal{S} \subseteq [n_1]$.

If all the output weights are negative, for example, the network expresses conjunction of linear inequalities. Therefore, we can learn a shallow rectifier network with negative output weights to indicate a system of linear inequalities. The network will predict positive if the system of linear equalities is satisfied. Using Corollary 6.2, we can covert the learned network into linear equalities, which can then be used in interesting ways, as we showed in Chapter 4.

Theorem 6.1 says we can use 2^{n_1} hyperplanes to construct the decision boundary in terms of a DNF expression. We demonstrate in Example 6.2 that not every one of those 2^{n_1} hyperplanes is always needed. In general, we know that at least one hyperplane is not needed—the one corresponding to $\mathcal{S}_+ = \mathcal{S}_- = \emptyset$. This is because $B(\mathbf{x}; \emptyset)$ is either always true or always false, depending on whether $b^{(2)} \geq 0$ or $b^{(2)} < 0$, and independent on the input $\mathbf{x}$. In fact, we know from Theorem 3.2 that there exist a shallow rectifier network whose decision boundary requires exactly $2^{n_1} - 1$ decision hyperplanes.

6.2.2 Deep rectifier networks

In this section, we extend the result of shallow rectifier network in Section 6.2.1 to the case of deep rectifier network with L hidden layers. We will develop a theorem, similar to Theorem 6.1, to capture the decision boundary of a deep rectifier network. We need the following definitions before we can state the main theorem.

6.2.2.1 Covering, path, and path value

Definition 6.6. *Given a feedforward rectifier network, a* covering $c^{(l)}$ *is a sequence of sets*

$$(\mathcal{S}^{(L)}, \mathcal{S}^{(L-1)}, \ldots, \mathcal{S}^{(l)})$$

where $\mathcal{S}^{(j)} \subseteq [n_j]$ *for* $j \in \{L, L-1, \ldots, l\}$.

Intuitively a covering $c^{(l)}$ selects, or *covers*, some units (possibly none) in each layer, starting from the L^{th} layer, going *backward* to the l^{th} layer. Figure 6.2 shows covering examples for a rectifier network with two hidden layers. We assume the output unit is

always covered and is not explicitly included in the definition of the covering. We also make the convention that for any covering $c^{(0)}$, $\mathcal{S}^{(0)} = [n_0]$; that is, all input units must be covered once the input layer is reached, as shown in Figure 6.2 (bottom).

A covering is related to the concept of activation pattern [91], since the latter is also defined in terms of subsets of hidden units. However, we differentiate a covering from the concept of the activation pattern. An activation pattern is defined based on a given input $\mathbf{x}$. Inputs from different linear regions will result in different activation patterns. Coverings, on the other hand, are defined for the network. We will see that a covering $c^{(0)}$ to the input layer specifies a hyperplane in the n_0-dimensional input space. This kind of hyperplanes will be the candidates for decision hyperplanes.

Next we define the concept of *a path* contained in a covering $c^{(l)}$.

Definition 6.7. *Given a covering $c^{(l)} = (\mathcal{S}^{(L)}, \mathcal{S}^{(L-1)}, \ldots, \mathcal{S}^{(l)})$, a path $p^{(l)}$ from the output unit to a unit in the l^{th} layer through the covering $c^{(l)}$ is a sequence of indices of the form $(i_L, i_{L-1}, \ldots, i_l)$, where $i_j \in \mathcal{S}^{(j)}$ for $j \in \{L, L-1, \ldots, l\}$. We write $p^{(l)} \sim c^{(l)}$ to indicate that the path $p^{(l)}$ is contained in the covering $c^{(l)}$. We also define the "* last*" operator to return the last element of a given path:* $\text{last}(i_L, i_{L-1}, \ldots, i_l) = i_l$.

Intuitively, a path selects one unit per layer *within* the covering, starting from the L^{th} layer to the l^{th} layer. The covering $c^{(l)}$ does not contain any paths if $\mathcal{S}^{(j)} = \emptyset$ for some layer $j \in \{L, \ldots, l\}$. Figure 6.2 illustrates all possible paths within three coverings. Note that the direction of a path is going backward: starting from the output unit, pointing toward the input layer.

Definition 6.8. *The value of a path $p^{(l)} = (i_L, i_{L-1}, \ldots, i_l)$ is defined as the product of the weight elements along the path:*

$$V(p^{(l)}) = V(i_L, i_{L-1}, \ldots, i_l) = W^{(L+1)}_{1,i_L} W^{(L)}_{i_L,i_{L-1}} \ldots W^{(l+1)}_{i_{l+1},i_l}$$

For example, in Figure 6.2 (bottom), the value of the path $(1, 2, 1)$ is $W^{(3)}_{11} W^{(2)}_{12} W^{(1)}_{21}$, and the value of the path $(2, 2, 1)$ is $W^{(3)}_{12} W^{(2)}_{22} W^{(1)}_{21}$.

Definition 6.9. *Given a covering $c^{(l+1)} = (S^{(L)}, S^{(L-1)}, \ldots, S^{(l+1)})$, we define the value of a unit $i_l \in [n_l]$ in the l^{th} layer as*

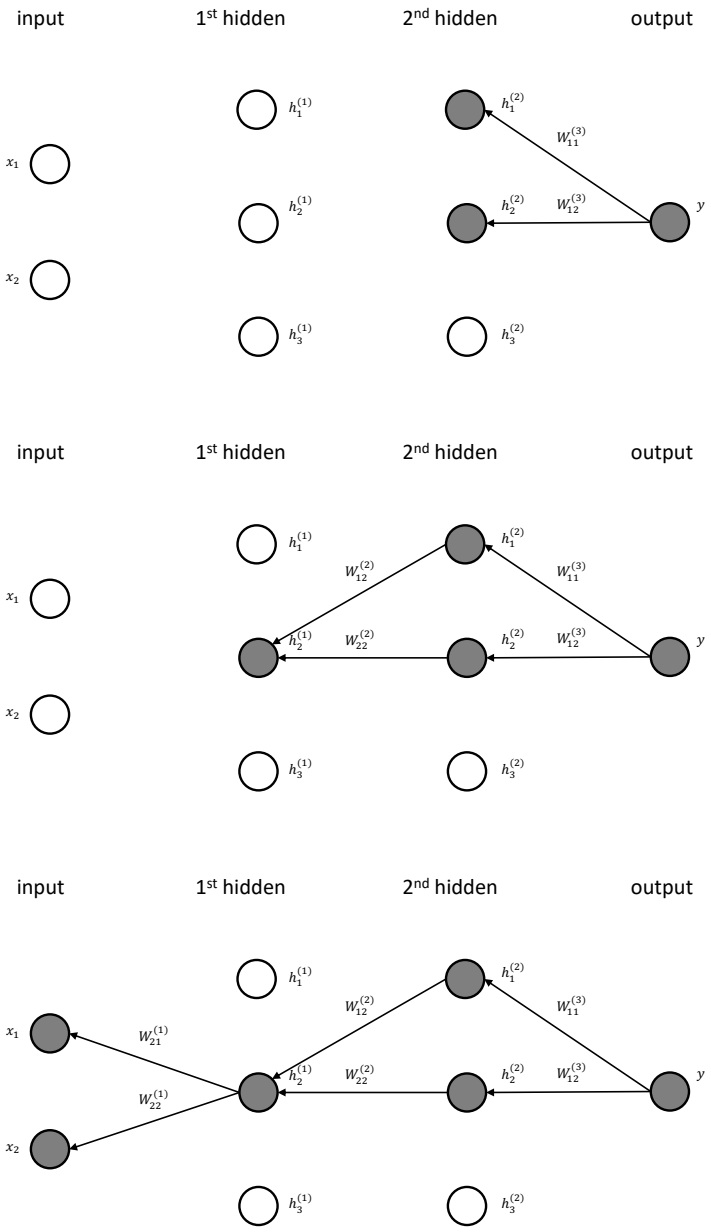

Figure 6.2. Illustration of three coverings: $c^{(2)}$, $c^{(1)}$, and $c^{(0)}$. Top: the covering $c^{(2)} = (\mathcal{S}^{(2)})$, where $\mathcal{S}^{(2)} = \{1, 2\}$ is shown in the shaded units in the 2nd hidden layer. Two paths contained in the covering $c^{(2)}$ are also shown. Middle: the covering $c^{(1)} = (\mathcal{S}^{(2)}, \mathcal{S}^{(1)})$, where $\mathcal{S}^{(1)} = \{2\}$. $c^{(1)}$ also contains two paths. Bottom: the covering $c^{(0)} = (\mathcal{S}^{(2)}, \mathcal{S}^{(1)}, \mathcal{S}^{(0)})$, where $\mathcal{S}^{(0)} = \{1, 2\}$. There are four paths contained in the covering $c^{(0)}$: $(1, 2, 1)$, $(1, 2, 2)$, $(2, 2, 1)$, and $(2, 2, 2)$. Note that in all cases the direction of the path is going backward.

$$V(i_l; c^{(l+1)}) = \sum_{i_L \in S^{(L)}} \sum_{i_{L-1} \in S^{(L-1)}} \cdots \sum_{i_{l+1} \in S^{(l+1)}} W^{(L+1)}_{1,i_L} W^{(L)}_{i_L,i_{L-1}} \cdots W^{(l+2)}_{i_{l+2},i_{l+1}} W^{(l+1)}_{i_{l+1},i_l}$$
$$= \sum_{p^{(l+1)} \sim c^{(l+1)}} V(p^{(l+1)}) W^{(l+1)}_{i_{l+1},i_l}$$

The summation in the last step is over all paths $p^{(l+1)}$ within the covering $c^{(l+1)}$, and i_{l+1} is the last element of $p^{(l+1)}$: $i_{l+1} = \text{last}(p^{(l+1)})$.

The value of a unit i_l in the l^{th} layer given a covering of $c^{(l+1)}$ is the sum of the values of all possible paths via the covering $c^{(l+1)}$ to the unit i_l. If $c^{(l+1)}$ contains no path, the value of the unit i_l is zero.

The value of a unit in the top layer is just $V(i_L) = W^{(L+1)}_{1,i_L}$. Note that when defining the value of the unit i_l in the l^{th} layer, the covering is only defined up to the $(l+1)^{\text{th}}$ layer. It is irrelevant whether the unit i_l is covered or not. For example, given the covering $c^{(2)}$ as shown in Figure 6.2 (top), the value of the first unit in the first hidden layer is $V(1; c^{(2)}) = W^{(3)}_{11} W^{(2)}_{11} + W^{(3)}_{12} W^{(2)}_{21}$. Just like in Definition 6.4 we define positive and negative units in shallow rectifier networks based on the weights, In a deep rectifier network, we will define positive and negative units in the l^{th} layer based on their values given the covering $c^{(l+1)}$.

6.2.2.2 Decision boundaries of deep rectifier networks

As in the shallow rectifier network case, we need to define positive and negative units for each hidden layer of a deep network. We start from the L^{th} layer, defining $\mathcal{U}^{(L)}_+$ and $\mathcal{U}^{(L)}_-$,

$$\mathcal{U}^{(L)}_+ = \{i_L \in [n_L] : V(i_L) > 0\}$$
$$\mathcal{U}^{(L)}_- = \{i_L \in [n_L] : V(i_L) < 0\}.$$

For the L^{th} hidden layer, $V(i_L) = W^{(L+1)}_{1,i_L}$, and therefore $\mathcal{U}^{(L)}_+$ contains those units in the last hidden layer with a positive output weight. Similarly $\mathcal{U}^{(L)}_-$ contains those with a negative output weight.

We use $\mathcal{S}^{(L)}_+$ (or $\mathcal{S}^{(L)}_-$) to refer a subset of $\mathcal{U}^{(L)}_+$ (or $\mathcal{U}^{(L)}_-$), and $\mathcal{S}^{(L)}$ is the union of $\mathcal{S}^{(L)}_+$ and $\mathcal{S}^{(L)}_-$:

$$\mathcal{S}^{(L)}_+ \subseteq \mathcal{U}^{(L)}_+, \quad \mathcal{S}^{(L)}_- \subseteq \mathcal{U}^{(L)}_-, \quad \mathcal{S}^{(L)} = \mathcal{S}^{(L)}_+ \cup \mathcal{S}^{(L)}_-.$$

Note that the choice of $\mathcal{S}^{(L)}$ leads to a covering $c^{(L)}$. Next for each $l \in [L-1]$, we define

$$\begin{aligned}\mathcal{U}_+^{(l)} &= \{i_l \in [n_l] : V(i_l; c^{(l+1)}) > 0\} \\ \mathcal{U}_-^{(l)} &= \{i_l \in [n_l] : V(i_l; c^{(l+1)}) < 0\}\end{aligned} \tag{6.8}$$

and

$$\mathcal{S}_+^{(l)} \subseteq \mathcal{U}_+^{(l)}, \quad \mathcal{S}_-^{(l)} \subseteq \mathcal{U}_-^{(l)}, \quad \mathcal{S}^{(l)} = \mathcal{S}_+^{(l)} \cup \mathcal{S}_-^{(l)}.$$

Once we pick (and fix) the subsets $\mathcal{S}_+^{(l)}$ and $\mathcal{S}_-^{(l)}$, we can append $\mathcal{S}^{(l)}$ to $c^{(l+1)}$ to obtain $c^{(l)}$ and iteratively define the positive and negative units for the $(l-1)^{\text{th}}$ layer. This process keeps going until we reach the input layer. Now we can define the hyperplanes that will be used to specify the decision boundary of the deep rectifier networks.

Definition 6.10. *Consider a deep rectifier network in Definition 6.1. For a sequence of sets $(\mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)})$ where $\mathcal{S}^{(l)} \subseteq [n_l]$, define a Boolean function $B(\mathbf{x}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)})$. The Boolean function $B(\mathbf{x}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)})$ takes the value true if the following linear inequality holds*

$$\sum_{p^{(0)} \sim c^{(0)}} V(p^{(0)}) x_{i_0} + \sum_{p^{(1)} \sim c^{(1)}} V(p^{(1)}) b_{i_1}^{(1)} + \cdots + \sum_{p^{(L)} \sim c^{(L)}} V(p^{(L)}) b_{i_L}^{(L)} + b^{(L+1)} \geq 0, \tag{6.9}$$

where $i_l = \text{last}(p^{(l)})$ for $l \in \{0, 1, \ldots, L\}$.

Similar to the case of shallow rectifier networks, the inequality in (6.9) is linear on $\mathbf{x}$, thus specifying a hyperplane in the input space. $B(\mathbf{x}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)})$ is true if $\mathbf{x}$ is at one side of the hyperplane and false if $\mathbf{x}$ is at the other side. The following theorem gives a construction of the decision boundary using these hyperplanes.

Theorem 6.2. *Consider the deep network given in Definition 6.1. For any input $\mathbf{x}$ we have the following equivalence conditions:*

$$y \geq 0 \iff \bigvee_{\mathcal{S}_+^{(L)} \subseteq \mathcal{U}_+^{(L)}} \bigwedge_{\mathcal{S}_-^{(L)} \subseteq \mathcal{U}_-^{(L)}} \cdots \bigvee_{\mathcal{S}_+^{(1)} \subseteq \mathcal{U}_+^{(1)}} \bigwedge_{\mathcal{S}_-^{(1)} \subseteq \mathcal{U}_-^{(1)}} B(\mathbf{x}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)}), \tag{6.10}$$

where $\mathcal{S}^{(l)} = \mathcal{S}_+^{(l)} \cup \mathcal{S}_-^{(l)}$ for $l \in [L]$, and the Boolean variable $B(\mathbf{x}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)})$ is true if the inequality in (6.9) *holds.*

Proof. We prove this theorem by induction on the number of hidden layers L.

Base case. In the base case in which $L = 1$, we can use Theorem 6.1 to get

$$y \geq 0 \iff \bigvee_{\mathcal{S}_+^{(1)} \subseteq \mathcal{U}_+^{(1)}} \bigwedge_{\mathcal{S}_-^{(1)} \subseteq \mathcal{U}_-^{(1)}} B(\mathbf{x}; \mathcal{S}^{(1)}),$$

where $B(\mathbf{x}; \mathcal{S}^{(1)})$ is true if the following inequality holds,

$$\sum_{i_1 \in \mathcal{S}^{(1)}} \sum_{i_0 \in [n_0]} W_{1,i_1}^{(2)} W_{i_1 i_0}^{(1)} x_{i_0} + \sum_{i_1 \in \mathcal{S}^{(1)}} W_{1,i_1}^{(2)} b_{i_1}^{(1)} + b^{(2)} \geq 0.$$

This is exactly the same as the inequality in (6.9) in the case of $L = 1$, since $c^{(1)} = (\mathcal{S}^{(1)})$ and $c^{(0)} = (\mathcal{S}^{(1)}, \mathcal{S}^{(0)})$ where $\mathcal{S}^{(0)} = [n_0]$.

Induction step. In the induction step we assume Theorem 6.2 holds for a network with $L - 1$ hidden layers, and we want to prove the equivalence condition in Eq. (6.17) for a network with L hidden layers. Notice that in Definition 6.1 the output y can be seen as a function of $\mathbf{h}^{(1)}$: $y = f(\mathbf{h}^{(1)})$, and this function $f(\mathbf{h}^{(1)})$ is a rectifier network with $(L-1)$ hidden layers, which has the same form as the original network defined in Definition 6.1. Therefore, we can apply the inductive hypothesis to this $(L-1)$-layer network to get the following equivalence condition:

$$y \geq 0 \iff \bigvee_{\mathcal{S}_+^{(L)} \subseteq \mathcal{U}_+^{(L)}} \bigwedge_{\mathcal{S}_-^{(L)} \subseteq \mathcal{U}_-^{(L)}} \cdots \bigvee_{\mathcal{S}_+^{(2)} \subseteq \mathcal{U}_+^{(2)}} \bigwedge_{\mathcal{S}_-^{(2)} \subseteq \mathcal{U}_-^{(2)}} B(\mathbf{h}^{(1)}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(2)}). \tag{6.11}$$

In the above equivalence condition, the Boolean variable $B(\mathbf{h}^{(1)}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(2)})$ is true if and only if

$$\sum_{\substack{p^{(2)} \sim c^{(2)} \\ i_1 \in [n_1]}} V(p^{(2)}) W_{i_2,i_1}^{(2)} h_{i_1}^{(1)} + \sum_{p^{(2)} \sim c^{(2)}} V(p^{(2)}) b_{i_2}^{(2)} + \cdots + \sum_{p^{(L)} \sim c^{(L)}} V(p^{(L)}) b_{i_L}^{(L)} + b^{(L+1)} \geq 0, \tag{6.12}$$

where $i_l = \text{last}(p^{(l)})$ for $l \in \{2, 3, \ldots, L\}$. Note that in the first term i_1 is summed over all units in the first hidden layer instead of $\mathcal{S}^{(1)}$ in order to apply the inductive hypothesis. This is because in the original definition of the covering $c^{(0)}$, we assume $\mathcal{S}^{(0)} = [n_0]$.

Using Definition 6.9, the first term in (6.12) can be written as

$$\sum_{\substack{p^{(2)} \sim c^{(2)} \\ i_1 \in [n_1]}} V(p^{(2)}) W_{i_2,i_1}^{(2)} h_{i_1}^{(1)} = \sum_{i_1 \in [n_1]} V(i_1; c^{(2)}) h_{i_1}^{(1)},$$

and we define b' to be the bias term in (6.12),

$$b' = \sum_{p^{(2)} \sim c^{(2)}} V(p^{(2)}) b_{i_2}^{(2)} + \cdots + \sum_{p^{(L)} \sim c^{(L)}} V(p^{(L)}) b_{i_L}^{(L)} + b^{(L+1)}.$$

The inequality in (6.12) can be rewritten as

$$\sum_{i_1 \in [n_1]} V(i_1; c^{(2)}) h_{i_1}^{(1)} + b' \geq 0. \tag{6.13}$$

Observe that the left side of (6.13) is the output of a rectifier network with one hidden layer, since $h_{i_1}^{(1)} = \text{rect}(z_{i_1}^{(1)}) = \text{rect}(\sum_{i_0 \in [n_0]} W_{i_1,i_0}^{(1)} x_{i_0} + b_{i_1}^{(1)})$. Therefore we can derive the equivalence condition of (6.13) using Theorem 6.1.

The inequality in (6.13) holds if and only if there exists a set $\mathcal{S}_+^{(1)} \subseteq \mathcal{U}_+^{(1)}$, such that for all set $\mathcal{S}_-^{(1)} \subseteq \mathcal{U}_-^{(1)}$, the following inequality holds

$$\sum_{i_1 \in \mathcal{S}^{(1)}} V(i_1; c^{(2)}) z_{i_1}^{(1)} + b' \geq 0, \tag{6.14}$$

where $\mathcal{S}^{(1)} = \mathcal{S}_+^{(1)} \cup \mathcal{S}_-^{(1)}$ and

$$\mathcal{U}_+^{(1)} = \{i_1 \in [n_1] : V(i_1; c^{(2)}) > 0\}$$
$$\mathcal{U}_-^{(1)} = \{i_1 \in [n_1] : V(i_1; c^{(2)}) < 0\}.$$

The inequality in (6.14) is equivalent to

$$\sum_{i_1 \in \mathcal{S}^{(1)}} \sum_{p^{(2)} \sim c^{(2)}} V(p^{(2)}) W_{i_2,i_1}^{(2)} \Big(\sum_{i_0 \in [n_0]} W_{i_1,i_0}^{(1)} x_{i_0} + b_{i_1}^{(1)} \Big) + b' \geq 0, \tag{6.15}$$

which can be rewritten as

$$\sum_{p^{(1)} \in P(c^{(1)})} V(p^{(1)}) \sum_{i_0 \in [n_0]} W_{i_1,i_0}^{(1)} x_{i_0} + \sum_{p^{(1)} \in P(c^{(1)})} V(p^{(1)}) b_{i_1}^{(1)} + b' \geq 0,$$

where $i_1 = \text{last}(p^{(1)})$. This is exactly the same condition as in (6.9); therefore, we have shown

$$B(\mathbf{h}^{(1)}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(2)}) = \bigvee_{\mathcal{S}_+^{(1)} \subseteq \mathcal{U}_+^{(1)}} \bigwedge_{\mathcal{S}_-^{(1)} \subseteq \mathcal{U}_-^{(1)}} B(\mathbf{x}; S^{(L)}, \ldots, \mathcal{S}^{(1)}). \tag{6.16}$$

Combining (6.11) and (6.16) gives the equivalence condition in Theorem 6.2. □

Here we use a simple two-layer rectifier network to illustrate how to obtain its decision boundary using Theorem 6.2.

Example 6.3. *Consider the following two layer rectifier network with two dimensional inputs:*

$$\begin{aligned} h_1^{(1)} &= \text{rect}(x_1) \\ h_2^{(1)} &= \text{rect}(x_1 + x_2) \\ h_1^{(2)} &= \text{rect}(h_1^{(1)}) \\ h_2^{(2)} &= \text{rect}(h_1^{(1)} - h_2^{(1)}) \\ y &= -h_1^{(2)} + h_2^{(2)}. \end{aligned}$$

This network has $\mathcal{U}_+^{(2)} = \{2\}$ *and* $\mathcal{U}_-^{(2)} = \{1\}$. *Therefore,* $\mathcal{S}_+^{(2)}$ *can be* $\emptyset$ *or* $\{2\}$, *and* $\mathcal{S}_-^{(2)}$ *can be* $\emptyset$ *or* $\{1\}$. *The first two columns in Table 6.1 specify the four choice of* $\mathcal{S}_+^{(2)}$ *and* $\mathcal{S}_-^{(2)}$. *For each choice of* $\mathcal{S}_+^{(2)}$ *and* $\mathcal{S}_-^{(2)}$, *we can determine* $\mathcal{U}_+^{(1)}$ *and* $\mathcal{U}_-^{(1)}$ *from Eq.* (6.8), *and the results are shown in the 3rd and 4th columns in Table 6.1. Now we can choose* $\mathcal{S}_+^{(1)} \subseteq \mathcal{U}_+^{(1)}$ *and* $\mathcal{S}_-^{(1)} \subseteq \mathcal{U}_-^{(1)}$ *and compute the corresponding hyperplane equations. See Table 6.1 for illustrations.*

The final decision boundary can be obtained by combining all the candidate hyperplanes in the last column of Table 6.1 using disjunctions and conjunctions, according to Eq. (6.17). *Here we only give the final result: the output* $y \geq 0$ *if and only if* $x_1 \leq 0$ *or* $x_1 + x_2 \leq 0$. *Note that just like in the shallow rectifier case, in this example, the final decision boundary is only specified by two decision hyperplanes, which is less than the number of possible candidate hyperplanes.*

Table 6.1. Determining hyperplanes from which the decision boundary of a two-layer rectifier network in Example 6.3 is made of. The table is filled column by column from left to right as we make choices for $\mathcal{S}_+^{(2)}$, $\mathcal{S}_-^{(2)}$, $\mathcal{S}_+^{(1)}$, and $\mathcal{S}_-^{(1)}$.

$\mathcal{S}_+^{(2)}$	$\mathcal{S}_-^{(2)}$	$\mathcal{U}_+^{(1)}$	$\mathcal{U}_-^{(1)}$	$\mathcal{S}_+^{(1)}$	$\mathcal{S}_-^{(1)}$	$B(\mathbf{x}; \mathcal{S}^{(2)}, \mathcal{S}^{(1)})$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$0 \geq 0$
$\emptyset$	$\{1\}$	$\emptyset$	$\{1\}$	$\emptyset$	$\emptyset$	$0 \geq 0$
				$\emptyset$	$\{1\}$	$-x_1 \geq 0$
$\{2\}$	$\emptyset$	$\{1\}$	$\{2\}$	$\emptyset$	$\emptyset$	$0 \geq 0$
				$\emptyset$	$\{2\}$	$-(x_1 + x_2) \geq 0$
				$\{1\}$	$\emptyset$	$x_1 \geq 0$
				$\{1\}$	$\{2\}$	$-x_2 \geq 0$
$\{2\}$	$\{1\}$	$\emptyset$	$\{2\}$	$\emptyset$	$\emptyset$	$0 \geq 0$
				$\emptyset$	$\{2\}$	$-(x_1 + x_2) \geq 0$

We can again consider the special case in which all the weights $W^{(l)}_{i_l,i_{l-1}}$ and $W^{(L+1)}_{1,i_l}$ are positive. In this case the negative set $\mathcal{U}^{(l)}_-$ is always the empty set for every layer $l \in [L]$, so is its subset $\mathcal{S}^{(l)}_-$. The decision $y \geq 0$ will be expressed in pure disjunctions and no conjunctions. We have the following corollary.

Corollary 6.3. *Consider the deep network given in Definition 6.1. If all the weights $W^{(l)}_{i_l,i_{l-1}}$ and $W^{(L+1)}_{1,i_l}$ are positive, then for any input $\mathbf{x}$ we have the following equivalence conditions:*

$$y \geq 0 \iff \bigvee_{\mathcal{S}^{(L)} \subseteq \mathcal{U}^{(L)}_+} \cdots \bigvee_{\mathcal{S}^{(1)} \subseteq \mathcal{U}^{(1)}_+} B(\mathbf{x}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)}), \tag{6.17}$$

where the Boolean variable $B(\mathbf{x}; \mathcal{S}^{(L)}, \ldots, \mathcal{S}^{(1)})$ is true if the inequality in (6.9) *holds.*

Note that in the case of all weights are negative, we do not have a similar corollary. We will still have positive and negative units alternating between adjacent layers.

6.2.3 Number of decision hyperplanes

In Theorem 6.2, we have shown that the decision boundary can be built from hyperplanes of the form in Eq. (6.9). Each hyperplane is parametrized by a sequence of sets $(S^{(L)}, \ldots, S^{(1)})$. We think of these hyperplanes as candidates for building the decision boundary. As we see in Example 6.3, the number of decision hyperplanes may be less, or even much less, than the number of candidate hyperplanes.

A trivial upper bound to the number of decision hyperplanes is just 2^N, where $N = \sum_{l=1}^{L} n_l$ is the total number of rectifier units in the network. This is because for each layer l, the number of choices for $\mathcal{S}^l$ is at most 2^{n_l}. We can also obtain this upper bound by using Proposition 6.1, with the fact that the number of linear regions is at most 2^N [64].

A slightly better upper bound to the number of candidate hyperplanes can be obtained by the following observation. If we want a candidate hyperplane specified by $(S^{(L)}, \ldots, S^{(1)})$ to be a decision hyperplane, then each set $\mathcal{S}^{(l)}$ must be nonempty for all layer l, because otherwise, the corresponding hyperplane in Eq. (6.9) will be input independent. Therefore, the number of decision hyperplanes is at most

$$\prod_{l \in [L]} (2^{n_l} - 1),$$

since there are at most $2^{n_l} - 1$ choices of nonempty set for $\mathcal{S}^{(l)}$. We emphasize that this upper bound is not tight in general. However, for shallow rectifier networks with large input dimensions, this is a tight bound as we show in Theorem 3.2.

6.3 Feedforward maxout networks

Our approach to construct the decision boundary of a feedforward rectifier network works equally well for maxout networks [33]. In this section, we discuss the decision boundary of maxout networks.

Definition 6.11. *The maxout activation function, when applied to a vector $\mathbf{z} \in \mathbb{R}^k$, is defined as*

$$\text{maxout}(\mathbf{z}) = \max\{z_1, z_2, \ldots, z_k\}. \tag{6.18}$$

That is, the maxout activation function returns the largest element of the given vector.

We consider the feedforward maxout neural networks given in the following definition.

Definition 6.12. *A feedforward maxout network with L hidden layers and one output unit is a function $F(\mathbf{x}; \theta) : \mathbb{R}^{n_0} \to \mathbb{R}$ in the following form:*

$$\begin{aligned} \mathbf{h}^{(0)} &= \mathbf{x} \\ \mathbf{z}^{(l)} &= \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \quad l \in [L] \\ \mathbf{h}^{(l)} &= \text{maxout}(\mathbf{z}^{(l)}), \quad l \in [L] \\ y &= \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + b^{(L+1)}, \end{aligned} \tag{6.19}$$

where $\mathbf{x} \in \mathbb{R}^{n_0}$ is the input to the network and $y \in \mathbb{R}$ is the output. $\mathbf{z}^{(l)}$ is the preactivation value of the l^{th} hidden layer, and $\mathbf{h}^{(l)}$ is the corresponding activated value. Assume this network has n_0 input units, n_l units in the l^{th} hidden layer, and one output unit. The parameters θ of the network are $\mathbf{W}^{(l)} \in \mathbb{R}^{k \cdot n_l \times n_{l-1}}$, $\mathbf{b}^{(l)} \in \mathbb{R}^{k \cdot n_l}$, $\mathbf{W}^{(L+1)} \in \mathbb{R}^{1 \times n_L}$, and $b^{(L+1)} \in \mathbb{R}$. Note that for all $l \in [L]$, $\mathbf{z}^{(l)} = [\mathbf{z}_1^{(l)}, \mathbf{z}_2^{(l)}, \ldots, \mathbf{z}_{n_l}^{(l)}]^\top \in \mathbb{R}^{k \cdot n_l}$, where each $\mathbf{z}_i^{(l)} \in \mathbb{R}^k$. The maxout function is applied "unit-wise" to $\mathbf{z}^{(l)}$:

$$\text{maxout}(\mathbf{z}^{(l)}) = [\text{maxout}(\mathbf{z}_1^{(l)}), \ldots, \text{maxout}(\mathbf{z}_{n_l}^{(l)})]^\top. \tag{6.20}$$

In order to use the feedforward maxout networks as classification functions, we again implicitly threshold the function at the value zero. Notice that maxout is also a piecewise

linear activation, so the function computed by the maxout network is again piecewise linear. We can still talk about linear regions and decision hyperplanes, and the relation between them in Proposition 6.1 still holds for maxout networks.

6.3.1 Shallow maxout networks

In this section, we derive the decision boundary of a maxout network with one hidden layer.

Definition 6.13. *A shallow maxout network is a function* $F(\mathbf{x};\theta) : \mathbb{R}^{n_0} \to \mathbb{R}$ *in the following form:*

$$\begin{aligned} \mathbf{z} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \mathbf{h} &= \text{maxout}(\mathbf{z}) \\ y &= \mathbf{W}^{(2)}\mathbf{h} + b^{(2)}, \end{aligned} \tag{6.21}$$

where $\mathbf{x} \in \mathbb{R}^{n_0}$ *is the input to the network, and* $y \in \mathbb{R}$ *is the output.* $\mathbf{z}$ *is the preactivation value of the hidden layer and* $\mathbf{h}$ *is the corresponding activated value. Assume this network has* n_1 *hidden units. The parameters* θ *of the network are* $\mathbf{W}^{(1)} \in \mathbb{R}^{k\cdot n_1 \times n_0}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{k\cdot n_1}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{1\times n_1}$, *and* $b^{(2)} \in \mathbb{R}$.

Given a shallow maxout network, again our goal is to specify a condition under which an input example $\mathbf{x}$ is classified as positive. We still define the set of *positive* and *negative units* in the hidden layer.

Definition 6.14. *Given a shallow maxout network in Definition 6.13, we define the set of positive and negative hidden units as*

$$\begin{aligned} \mathcal{U}_+ &= \{i_1 \in [n_1] : W^{(2)}_{1,i_1} > 0\}, \\ \mathcal{U}_- &= \{i_1 \in [n_1] : W^{(2)}_{1,i_1} < 0\}. \end{aligned}$$

$\mathcal{U}_+$ contains the indices of the hidden units whose weights are positive, and $\mathcal{U}_-$ contains those with negative weights.

6.3.1.1 Coloring

In order to describe the decision boundary of a rectifier network, we consider subsets of hidden units. This is because the rectifier function $\text{rect}(\cdot)$ has two branches, one of which

is always zero. For maxout network, each maxout unit has k operating modes, therefore we can think about coloring the maxout units with k different colors.

Suppose each maxout unit $h_i^{(1)}$ can be colored with k different colors. We use integers 1 through k to label these colors. We define the coloring c as follows.

Definition 6.15. *A* coloring c *is a tuple* $(j_1, j_2, \ldots, j_{n_1}) \in [k]^{n_1}$ *where each* $j_i \in [k]$. *Thus a coloring specify a color for every maxout unit in the hidden layer.*

We use the notation $c_{i_1} = j_{i_1}$ to indicate that, under the coloring c, the maxout unit i_1 is colored as j_{i_1}. Since $W^{(1)} \in \mathbb{R}^{k \cdot n_1 \times n_0}$ we need three indices to index an element in $W^{(1)}$. A generic element of $W^{(1)}$ is written as $W^c_{i_1,i_0}$, where i_0 and i_1 specify the unit in the input layer and the hidden layer, respectively, and the color of the unit i_1 is understood to be specified by coloring c as c_{i_1}. Similarly an element of $b^{(1)}$ is written as $b^c_{i_1}$.

A coloring c induces a coloring c_+ on the positive units in $\mathcal{U}_+$, and a coloring c_- on the negative units in $\mathcal{U}_-$. Similarly, given a positive coloring c_+ and a negative coloring c_-, we can infer a coloring c on the entire set of units in the hidden layer. For those units i_1 such that $W^{(2)}_{1,i_1} = 0$, we can arbitrarily assign a color to it, say $j_{i_1} = 1$. We will see that the colors of such units do not affect our analysis. Therefore, we have a bidirectional one to one mapping between c and (c_+, c_-). We write this relation as $c = c_+ + c_-$.

6.3.1.2 Decision boundary of shallow maxout networks

Definition 6.16. *Given a coloring* c, *we define a Boolean variable* $B(\mathbf{x}; c)$. $B(\mathbf{x}; c)$ *is true if the following linear inequality holds*

$$\sum_{i_1 \in [n_1]} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^c_{i_1,i_0} x_{i_0} + \sum_{i_1 \in [n_1]} W^{(2)}_{1,i_1} b^c_{i_1} + b^{(2)} \geq 0. \tag{6.22}$$

The only differences between inequality in (6.22) and the one in (6.3) are that now we need a coloring c in the superscripts, and the sum of i_1 is over all hidden units.

Now we are ready to state the theorem on decision boundary of the shallow maxout network.

Theorem 6.3. *Consider the shallow maxout network given in Definition 6.13. For any input* $\mathbf{x}$, *the corresponding output* $y \geq 0$ *if and only if there exists a coloring* c_+ *on* $\mathcal{U}_+$ *such that for any coloring* c_- *on* $\mathcal{U}_-$, *the value of* $B(\mathbf{x}; c)$ *is true, where* $c = c_+ + c_-$. *Writing in logic form we have*

$$y \geq 0 \iff \bigvee_{c_+} \bigwedge_{c_-} B(\mathbf{x}; c).$$

Proof. Let us first prove the forward direction of the implication. Assuming $y \geq 0$, we want to show that there exist a coloring c_+ such that for every possible coloring c_-, the Boolean variable $B(\mathbf{x}; c)$ is true. We can construct the coloring c_+ in the following way. For every maxout unit $i_1 \in \mathcal{U}_+$, pick the color of this unit as

$$c_{+,i_1} = \arg\max_{j \in [k]} z_{i_1,j}. \tag{6.23}$$

Note that $\mathbf{z}_{i_1} \in \mathbb{R}^k$ has k preactivation components, and c_+ assign color to unit i_1 based on the maximizer of these k components. Such constructed coloring c_+ has the desired property. To see that, from $y \geq 0$ we have

$$\begin{aligned}
0 &\leq \sum_{i_1 \in [n_1]} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + b^{(2)} && (6.24)\\
&= \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + b^{(2)} && (6.25)\\
&= \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} \Big(\sum_{i_0 \in [n_0]} W^{c_+}_{i_1,i_0} x_{i_0} + b^{c_+}_{i_1} \Big) + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + b^{(2)} && (6.26)\\
&= \sum_{i_1 \in \mathcal{U}_+} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_+}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} b^{c_+}_{i_1} + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + b^{(2)}. && (6.27)
\end{aligned}$$

Also we know that for any unit $i_1 \in \mathcal{U}_-$ and any coloring c_-,

$$\operatorname{maxout}(\mathbf{z}_{i_1}) \geq \sum_{i_0 \in [n_0]} W^{c_-}_{i_1,i_0} x_{i_0} + b^{c_-}_{i_1}. \tag{6.28}$$

Therefore,

$$\sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) \leq \sum_{i_1 \in \mathcal{U}_-} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_-}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} b^{c_-}_{i_1}. \tag{6.29}$$

Combining inequalities in (6.27) and (6.29),

$$\begin{aligned}
&\sum_{i_1 \in \mathcal{U}_+} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_+}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} b^{c_+}_{i_1}\\
+ &\sum_{i_1 \in \mathcal{U}_-} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_-}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} b^{c_-}_{i_1} + b^{(2)} \geq 0,
\end{aligned} \tag{6.30}$$

which is equivalent to

$$\sum_{i_1 \in [n_1]} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in [n_1]} W^{(2)}_{1,i_1} b^{c}_{i_1} + b^{(2)} \geq 0. \tag{6.31}$$

This concludes the proof of the forward direction of the implication in the theorem.

Next we prove the backward direction of the implication. We know that there exist a particular coloring c_+, such that for any coloring c_-, $B(\mathbf{x}; c)$ is true. In particular, $B(\mathbf{x}; c)$ is true for a coloring c_- defined in the following way. For every maxout unit $i_1 \in \mathcal{U}_-$, pick the color of this unit as

$$c_{-,i_1} = \arg\max_{j \in [k]} z_{i_1,j}. \tag{6.32}$$

For this particular coloring c_-, $B(\mathbf{x}; c)$ is true means

$$\begin{aligned} &\sum_{i_1 \in \mathcal{U}_+} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_+}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} b^{c_+}_{i_1} \\ &+ \sum_{i_1 \in \mathcal{U}_-} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_-}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} b^{c_-}_{i_1} + b^{(2)} \geq 0. \end{aligned} \tag{6.33}$$

Note that

$$\begin{aligned} \sum_{i_1 \in \mathcal{U}_+} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_+}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} b^{c_+}_{i_1} &= \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} \Big(\sum_{i_0 \in [n_0]} W^{c_+}_{i_1,i_0} x_{i_0} + b^{c_+}_{i_1} \Big) \\ &\leq \sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) \end{aligned} \tag{6.34}$$

and

$$\begin{aligned} \sum_{i_1 \in \mathcal{U}_-} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c_-}_{i_1,i_0} x_{i_0} + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} b^{c_-}_{i_1} &= \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} \Big(\sum_{i_0 \in [n_0]} W^{c_-}_{i_1,i_0} x_{i_0} + b^{c_-}_{i_1} \Big) \\ &= \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}). \end{aligned} \tag{6.35}$$

Combining inequalities in (6.33), (6.34) and (6.35)

$$\sum_{i_1 \in \mathcal{U}_+} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + \sum_{i_1 \in \mathcal{U}_-} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + b^{(2)} \geq 0, \tag{6.36}$$

which is equivalent to

$$\sum_{i_1 \in [n_1]} W^{(2)}_{1,i_1} \operatorname{maxout}(\mathbf{z}_{i_1}) + b^{(2)} \geq 0$$

or

$$y \geq 0.$$

□

6.3.2 Deep maxout networks

The goal of this section is to derive the decision boundary for a deep maxout network. Analogous to the concept of *covering* defined in Definition 6.7, we begin by defining a *palette*, which is the corresponding concept for deep maxout networks.

6.3.2.1 Palette and colored path

Definition 6.17. *A palette $t^{(l)}$ is a sequence of coloring $(c^{(L)}, c^{(L-1)}, \ldots, c^{(l)})$.*

Intuitively a palette colors every units starting from the L^{th} layer, going backward to the l^{th} layer.

Definition 6.18. *Given a palette $t^{(l)} = (c^{(L)}, c^{(L-1)}, \ldots, c^{(l)})$, a colored path $p^{(l)}$ from the L^{th} layer to the l^{th} layer is a sequence of pairs $((i_L, j_L), (i_{L-1}, j_{L-1}), \ldots, (i_l, j_l))$, where $i_m \in [n_m]$ specifies a unit in the m^{th} layer for $m \in \{L, L-1, \ldots, l\}$, and $j_m \in [k]$ is the color of the unit i_m according to the palette $t^{(l)}$; that is, $j_m = c^{(m)}_{i_m}$. We indicate the colored path $p^{(l)}$ is defined on the palette $t^{(l)}$ by writing $p^{(l)} \sim t^{(l)}$. We also define the "* last*" operator to return the last unit of a colored path:* $\text{last}((i_L, j_L), (i_{L-1}, j_{L-1}), \ldots, (i_l, j_l)) = (i_l, j_l)$.

Definition 6.19. *Given a colored path $p^{(l)} = ((i_L, j_L), (i_{L-1}, j_{L-1}), \ldots, (i_l, j_l))$, we define its value as*

$$\begin{aligned} V(p^{(l)}) &= V((i_L, j_L), \ldots, (i_l, j_l)) \\ &= W_{1,i_L} W^{j_L}_{i_L, i_{L-1}} \cdots W^{j_{l+1}}_{i_{l+1}, i_l}. \end{aligned}$$

Note that we omit the layer indices on the superscripts of W's since they can be inferred from the subscript indices. The value of a colored path does not depend on the color of its last unit j_l.

Definition 6.20. *Given a palette $t^{(l+1)} = (c^{(L)}, c^{(L-1)}, \ldots, c^{(l+1)})$, the value of a unit i_l in the l^{th} layer is*

$$\begin{aligned} V(i_l; t^{(l+1)}) &= V(i_l; c^{(L)}, c^{(L-1)}, \ldots, c^{(l+1)}) \\ &= \sum_{i_L \in [n_L]} \sum_{i_{L-1} \in [n_{L-1}]} \cdots \sum_{i_{l+1} \in [n_{l+1}]} W^{(L+1)}_{1,i_L} W^{c^{(L)}}_{i_L, i_{L-1}} \cdots W^{c^{(l+2)}}_{i_{l+2}, i_{l+1}} W^{c^{(l+1)}}_{i_{l+1}, i_l} \\ &= \sum_{p^{(l)} \sim t^{(l+1)}} V(p^{(l+1)}) W^{c^{(l+1)}}_{i_{l+1}, i_l} \end{aligned}$$

The summation in the last step is over all colored paths $p^{(l+1)}$ defined on the palette $t^{(l+1)}$, and i_{l+1} is the last element of $p^{(l+1)}$: $i_{l+1} = \text{last}(p^{(l+1)})$.

Therefore, the value of a unit i_l in the l^{th} layer given a palette of $t^{(l+1)}$ is the sum of the values of all possible paths via the palette $t^{(l+1)}$ to the unit i_l. Note that the value

of a unit does not depend on its own color. The value of a unit in the top layer is just $V(i_L) = W^{(L+1)}_{1,i_L}$.

6.3.2.2 Decision boundaries of deep maxout networks

We recursively define a series of sets of positive and negative units for each layer. We start from the L^{th} layer, defining $\mathcal{U}^{(L)}_+$ and $\mathcal{U}^{(L)}_-$,

$$\mathcal{U}^{(L)}_+ = \{i_L \in [n_L] : V(i_L) > 0\}$$
$$\mathcal{U}^{(L)}_- = \{i_L \in [n_L] : V(i_L) < 0\}.$$

We use $c^{(L)}_+$ (or $c^{(L)}_-$) to refer a coloring on $\mathcal{U}^{(L)}_+$ (or $\mathcal{U}^{(L)}_-$), and $c^{(L)}$ to refer the combined coloring of $c^{(L)}_+$ and $c^{(L)}_-$ on every units in $[n_L]$, that is, $c^{(L)} = c^{(L)}_+ + c^{(L)}_-$. As we did in the one-hidden layer case, the color of a unit i_L with zero value of $V(i_L)$ could be arbitrary. Note that the choice of $c^{(L)}$ leads to a palette $t^{(L)}$. Next for each $l \in [L-1]$ we define

$$\mathcal{U}^{(l)}_+ = \{i_l \in [n_l] : V(i_l; t^{(l+1)}) > 0\}$$
$$\mathcal{U}^{(l)}_- = \{i_l \in [n_l] : V(i_l; t^{(l+1)}) < 0\}$$

and use $c^{(l)}_+$ and $c^{(l)}_-$ to refer a coloring on $\mathcal{U}^{(l)}_+$ and $\mathcal{U}^{(l)}_-$, respectively. We then combine $c^{(l)}_+$ and $c^{(l)}_-$ to get $c^{(l)}$ and append it to $t^{(l+1)}$ to obtain $t^{(l)}$. This process keeps going until reaching the input layer.

Definition 6.21. *Consider a deep maxout network in Definition 6.12. For a sequence of coloring $(c^{(L)}, \ldots, c^{(1)})$, define a Boolean function $B(\mathbf{x}; c^{(L)}, \ldots, c^{(1)})$. The Boolean function $B(\mathbf{x}; c^{(L)}, \ldots, c^{(1)})$ takes the value true if the following linear inequality holds*

$$\sum_{\substack{p^{(1)} \sim t^{(1)} \\ i_0 \in [n_0]}} V(p^{(1)}) W^{j_1}_{i_1,i_0} x_{i_0} + \sum_{p^{(1)} \sim t^{(1)}} V(p^{(1)}) b^{j_1}_{i_1} + \cdots + \sum_{p^{(L)} \sim t^{(L)}} V(p^{(L)}) b^{j_L}_{i_L} + b^{(L+1)} \geq 0 \quad (6.37)$$

where $(i_l, j_l) = \text{last}(p^{(l)})$ for $l \in [L]$.

The inequality in (6.37) is linear on $\mathbf{x}$ and represents a hyperplane in the input space. The decision boundary of a deep maxout network can be expressed using these hyperplanes, as shown in Theorem 6.4.

Theorem 6.4. *Consider the deep maxout network in Definition 6.12. For any input x we have the following equivalence conditions:*

$$y \geq 0 \iff \bigvee_{c_+^{(L)}} \bigwedge_{c_-^{(L)}} \cdots \bigvee_{c_+^{(1)}} \bigwedge_{c_-^{(1)}} B(\mathbf{x}; c^{(L)}, \ldots, c^{(1)}) \tag{6.38}$$

where the Boolean variable $B(\mathbf{x}; c^{(L)}, \ldots, c^{(1)})$ is true if the inequality in (6.37) *holds.*

Proof. We prove this theorem by induction on the number of hidden layers.

Base case. In the base case in which $L = 1$, we can use Theorem 6.3 to get

$$y \geq 0 \iff \bigvee_{c_+^{(1)}} \bigwedge_{c_-^{(1)}} B(\mathbf{x}; c^{(1)}),$$

where $B(\mathbf{x}; c^{(1)})$ is true if and only if the following inequality holds,

$$\sum_{i_1 \in [n_1]} \sum_{i_0 \in [n_0]} W^{(2)}_{1,i_1} W^{c^{(1)}}_{i_1 i_0} x_{i_0} + \sum_{i_1 \in [n_1]} W^{(2)}_{1,i_1} b^{c^{(1)}}_{i_1} + b^{(2)} \geq 0.$$

This is exactly the same as Eq. (6.37) in the case of $L = 1$.

Induction step. In the induction step we assume Theorem 6.4 holds for a network with $L - 1$ hidden layers, and we want to prove the equivalence condition in Eq. (6.38) for a network with L hidden layers. Notice that in Eq. (6.19) the output y can be seen as a function of $\mathbf{h}^{(1)}$: $y = f(\mathbf{h}^{(1)})$, and this function $f(\cdot)$ is an $(L-1)$-layer maxout network, which has the same form as the original network in Definition 6.12. Therefore, we can apply the inductive hypothesis to this $(L-1)$-layer network to get the following equivalence condition:

$$y \geq 0 \iff \bigvee_{c_+^{(L)}} \bigwedge_{c_-^{(L)}} \cdots \bigvee_{c_+^{(2)}} \bigwedge_{c_-^{(2)}} B(\mathbf{h}^{(1)}; c^{(L)}, \ldots, c^{(2)}). \tag{6.39}$$

In the above equivalence condition, the Boolean variable $B(\mathbf{h}^{(1)}; c^{(L)}, \ldots, c^{(2)})$ is true if and only if

$$\sum_{\substack{p^{(2)} \sim t^{(2)} \\ i_1 \in [n_1]}} V(p^{(2)}) W^{j_2}_{i_2, i_1} h^{(1)}_{i_1} + \sum_{p^{(2)} \sim t^{(2)}} V(p^{(2)}) b^{j_2}_{i_2} + \cdots + \sum_{p^{(L)} \sim t^{(L)}} V(p^{(L)}) b^{j_L}_{i_L} + b^{(L+1)} \geq 0, \tag{6.40}$$

where $(i_l, j_l) = \text{last}(p^{(l)})$ for $l \in \{2, 3, \ldots, L\}$. Note that the first term in Eq. (6.40) is

$$\sum_{\substack{p^{(2)} \sim t^{(2)} \\ i_1 \in [n_1]}} V(p^{(2)}) W^{j_2}_{i_2, i_1} h^{(1)}_{i_1} = \sum_{i_1 \in [n_1]} V(i_1; t^{(2)}) h^{(1)}_{i_1},$$

and we define b' to be the bias term in (6.40),

$$b' = \sum_{p^{(2)} \sim t^{(2)}} V(p^{(2)}) b_{i_2}^{j_2} + \cdots + \sum_{p^{(L)} \sim t^{(L)}} V(p^{(L)}) b_{i_L}^{j_L} + b^{(L+1)}.$$

Equation (6.40) can be rewritten as

$$\sum_{i_1 \in [n_1]} V(i_1; t^{(2)}) h_{i_1}^{(1)} + b' \geq 0. \tag{6.41}$$

Now we can define

$$\mathcal{U}_+^{(1)} = \{i_1 \in [n_1] : V(i_1; t^{(2)}) > 0\}$$
$$\mathcal{U}_-^{(1)} = \{i_1 \in [n_1] : V(i_1; t^{(2)}) < 0\},$$

and the coloring $c_+^{(1)}$ and $c_-^{(1)}$. Using Theorem 6.3, the inequality in Eq. (6.41) holds if and only if there exists a coloring $c_+^{(1)}$, such that for all coloring $c_-^{(1)}$, the following inequality holds

$$\sum_{i_1 \in [n_1]} V(i_1; t^{(2)}) z_{i_1}^{c^{(1)}} + b' \geq 0,$$

which is equivalent to

$$\sum_{i_1 \in [n_1]} \sum_{p^{(2)} \sim t^{(2)}} V(p^{(2)}) W_{i_2, i_1}^{j_2} \Big(\sum_{i_0 \in [n_0]} W_{i_1, i_0}^{c^{(1)}} x_{i_0} + b_{i_1}^{c^{(1)}} \Big) + b' \geq 0, \tag{6.42}$$

where $(i_2, j_2) = \text{last}(p^{(2)})$. Equation (6.42) can be rewritten as

$$\sum_{p^{(1)} \sim t^{(1)}} V(p^{(1)}) \sum_{i_0 \in [n_0]} W_{i_1, i_0}^{j_1} x_{i_0} + \sum_{p^{(1)} \sim t^{(1)}} V(p^{(1)}) b_{i_1}^{j_1} + b' \geq 0,$$

where $(i_1, j_1) = \text{last}(p^{(1)})$. This is exactly the same condition as in Eq. (6.37); therefore, we have shown

$$B(\mathbf{h}^{(1)}; c^{(L)}, \ldots, c^{(2)}) = \bigvee_{c_+^{(1)}} \bigwedge_{c_-^{(1)}} B(\mathbf{x}; c^{(L)}, \ldots, c^{(1)}). \tag{6.43}$$

Combining Eq. (6.39) and Eq. (6.43) gives the equivalence condition in Theorem 6.4. □

6.3.3 Number of hyperplanes for maxout networks

In this section, we count the number of hyperplanes needed to specify the decision boundary of the network in Eq. (6.19). This requires us to count the number of palette $t^{(1)}$, or the number of distinct coloring sequences $(c^{(L)}, \ldots, c^{(1)})$, since each choice leads to a

hyperplane. The number of choices of $c^{(l)}$ is k^{n_l}, since there are k colors to choose from for each unit in the l^{th} layer. Therefore the maximum number of hyperplanes needed to specify the decision boundary of the network in Eq. (6.19) is

$$T(n_1, \ldots, n_L) = \prod_{l \in [L]} k^{n_l} = k^{\sum_{l \in [L]} n_l} = k^N,$$

where N is the total number of maxout units.

6.4 Conclusions and outlook

In this chapter, we provide a systematic way of constructing the decision boundary of deep neural networks with piecewise linear activations. We use rectifier and maxout networks as examples to illustrate under what condition a given input will be labeled as positive by the network.

The decision boundaries of such networks are composed of a series of candidate hyperplanes. We identify these hyperplanes parameterized by a sequence of subsets of units, in the case of rectifier networks, and by a sequence of colored units, in the case of maxout networks. The maximum number of such candidate hyperplanes can be exponential in the number of hidden units in the network. We provide a clean way of combining candidate hyperplanes to obtain the decision boundaries. In general, not every candidate hyperplane is needed to specify the decision boundary, and the number of decision hyperplanes can be much less than the number of candidate hyperplanes. An interesting future research direction is to study how to detect the actual decision hyperplanes from a large number of candidate hyperplanes.

We believe that our approach of analyzing the decision boundaries of piecewise linear neural networks complement the works on the linear regions of such networks. The mathematical construction in our analysis can be used to study the expressiveness of neural networks and to understand their empirical successes in many machine learning tasks.

CHAPTER 7

CONCLUSIONS AND FUTURE DIRECTIONS

In this dissertation, I study the constraints in structured prediction problems. Constraints are essential for correctly solving structured prediction problems. However, obtaining constraints is not an easy task, often requiring domain expertise. I provide an approach to automatically discover constraints from the training data of structured prediction problems. The approach is based on a theoretical study on the expressiveness of shallow rectifier networks.

Predicting a structured output in the presence of complicated constraints is a computationally hard problem because the output space is combinatorially large, and the output structure is discrete. I propose a systematic approach to learn a speedup classifier to imitate a given structure classifier. The goal of the speedup classifier is to reduce the inference time while maintaining the accuracy of the original structure classifier. To achieve this, I frame the structured prediction problems as search problems and learn heuristic functions to speedup the inference process.

In the last part of the dissertation, I extend the results on the expressiveness of shallow rectifier networks to deep neural networks with piecewise linear activation functions. I provide a way of constructing the decision boundaries for such networks. I show that the decision boundaries are composed of a system of hyperplanes in a unique manner.

There are of cause many future directions worth pursuing along the line of this dissertation. I list some of them as follows:

1. Just like shallow networks, the decision boundaries of the deep neural networks are composed of a system of hyperplanes. Can we use deep networks to learn linear inequality constraints for structured prediction problems? If we can, does using deep networks give us better results compared with using shallow ones?

2. What is the best way to generate negative examples for constraint learning problems?

We have shown that one way to generate negative examples is to perturb the positive examples. Is there any better way of doing this?

3. How to interpret the learned constraints? How to reduce redundancies in the learned constraints.

4. When solving structured prediction problems with learning to search approach, can we determine in an online fashion that whether we should use the trained classifier, or use the speedup classifier?

I believe the above questions are important and whose solutions can further benefit the fields such as structured prediction, natural language processing, and deep learning.

REFERENCES

[1] S. Anzaroot, A. Passos, D. Belanger, and A. McCallum, *Learning soft linear constraints with application to citation field extraction*, in Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Baltimore, Maryland, June 2014, Association for Computational Linguistics, pp. 593–602.

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

[3] J. Berant and P. Liang, *Imitation learning of agenda-based semantic parsers*, Transactions of the Association for Computational Linguistics, 3 (2015), pp. 545–558.

[4] A. Blum and R. L. Rivest, *Training a 3-node neural network is np-complete*, in Advances in Neural Information Processing Systems 1, D. S. Touretzky, ed., Morgan-Kaufmann, 1989, pp. 494–501.

[5] N. Bodenstab, A. Dunlop, K. Hall, and B. Roark, *Beam-width prediction for efficient context-free parsing*, in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, Oregon, USA, June 2011, Association for Computational Linguistics, pp. 440–449.

[6] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, *Model compression*, in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, New York, NY, USA, 2006, ACM, pp. 535–541.

[7] K.-W. Chang, A. Krishnamurthy, A. Agarwal, H. Daume, and J. Langford, *Learning to search better than your teacher*, in Proceedings of the 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings of Machine Learning Research, Lille, France, 07–09 Jul 2015, PMLR, pp. 2058–2066.

[8] M.-W. Chang, L. Ratinov, and D. Roth, *Guiding semi-supervision with constraint-driven learning*, in Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Prague, Czech Republic, June 2007, Association for Computational Linguistics, pp. 280–287.

[9] ——, *Structured learning with constrained conditional models*, Machine Learning, 88 (2012), pp. 399–431.

[10] Y.-W. Chang and M. Collins, *Exact decoding of phrase-based translation models through Lagrangian relaxation*, in Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, Edinburgh, Scotland, UK., July 2011, Association for Computational Linguistics, pp. 26–37.

[11] C. Chow and C. Liu, *Approximating discrete probability distributions with dependence trees*, IEEE Transactions on Information Theory, 14 (1968), pp. 462–467.

[12] W. W. Cohen and Y. Singer, *A simple, fast, and effictive rule learner*, in Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA, 1999, pp. 335–342.

[13] M. Collins, *Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms*, in Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002), Association for Computational Linguistics, July 2002, pp. 1–8.

[14] M. Collins and B. Roark, *Incremental parsing with the perceptron algorithm*, in Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04), Barcelona, Spain, July 2004, pp. 111–118.

[15] G. F. Cooper and E. Herskovits, *A bayesian method for the induction of probabilistic networks from data*, Machine Learning, 9 (1992), pp. 309–347.

[16] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals and Systems, 2 (1989), pp. 303–314.

[17] A. Daniely, N. Linial, and S. Shalev-Shwartz, *From average case complexity to improper learning complexity*, in Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing, STOC '14, New York, NY, USA, 2014, ACM, pp. 441–448.

[18] D. Das, A. F. T. Martins, and N. A. Smith, *An exact dual decomposition algorithm for shallow semantic parsing with constraints*, in *SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012), Montréal, Canada, 7-8 June 2012, Association for Computational Linguistics, pp. 209–217.

[19] B. DasGupta and G. Schnitger, *The power of approximating: a comparison of activation functions*, in Advances in Neural Information Processing Systems 5, S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., Morgan-Kaufmann, 1993, pp. 615–622.

[20] H. Daumé, J. Langford, and D. Marcu, *Search-based structured prediction*, Machine Learning, 75 (2009), pp. 297–325.

[21] H. Daumé, III and D. Marcu, *Learning as search optimization: Approximate large margin methods for structured prediction*, in Proceedings of the 22Nd International Conference on Machine Learning, ICML '05, New York, NY, USA, 2005, ACM, pp. 169–176.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, June 2019, Association for Computational Linguistics, pp. 4171–4186.

[23] J. R. Doppa, A. Fern, and P. Tadepalli, *Hc-search: A learning framework for search-based structured prediction*, J. Artif. Intell. Res., 50 (2014), pp. 369–407.

[24] G. Durrett and D. Klein, *Neural CRF parsing*, in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Beijing, China, July 2015, Association for Computational Linguistics, pp. 302–312.

[25] A. Fern, *Speedup learning*, in Encyclopedia of Machine Learning and Data Mining, C. Sammut and G. I. Webb, eds., Springer US, Boston, MA, 2017, pp. 1167–1172.

[26] J. R. Finkel, T. Grenager, and C. Manning, *Incorporating non-local information into information extraction systems by Gibbs sampling*, in Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), Ann Arbor, Michigan, June 2005, Association for Computational Linguistics, pp. 363–370.

[27] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, *Learning Probabilistic Relational Models*, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 2 (1999), pp. 1300–1307.

[28] L. Getoor, *Learning Statistical Models from Relational Data*, PhD thesis, Stanford University, Stanford, California, USA, 2001.

[29] L. Getoor and L. Mihalkova, *Learning statistical models from relational data*, in Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, New York, NY, USA, 2011, ACM, pp. 1195–1198.

[30] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, G. Gordon, D. Dunson, and M. Dudík, eds., vol. 15 of Proceedings of Machine Learning Research, Fort Lauderdale, FL, USA, 11–13 Apr 2011, PMLR, pp. 315–323.

[31] Y. Goldberg, *Neural Network Methods for Natural Language Processing*, Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers, 2017.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, Massachusetts, 2016.

[33] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, *Maxout networks*, in Proceedings of the 30th International Conference on Machine Learning, S. Dasgupta and D. McAllester, eds., vol. 28 of Proceedings of Machine Learning Research, Atlanta, Georgia, USA, 17–19 Jun 2013, PMLR, pp. 1319–1327.

[34] D. Graff, J. Kong, K. Chen, and K. Maeda, *English gigaword*, Linguistic Data Consortium, Philadelphia, 4 (2003), p. 1.

[35] Gurobi Optimization LLC, *Gurobi optimizer reference manual*, 2019.

[36] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, *Threshold circuits of bounded depth*, J. Comput. Syst. Sci., 46 (1993), pp. 129–154.

[37] H. He, H. Daumé III, and J. Eisner, *Dynamic feature selection for dependency parsing*, in Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Seattle, Washington, USA, Oct. 2013, Association for Computational Linguistics, pp. 1455–1464.

[38] H. He, H. Daume III, and J. M. Eisner, *Learning to search in branch and bound algorithms*, in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., Curran Associates, Inc., 2014, pp. 3293–3301.

[39] L. He, K. Lee, M. Lewis, and L. Zettlemoyer, *Deep semantic role labeling: What works and what's next*, in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, Canada, July 2017, Association for Computational Linguistics, pp. 473–483.

[40] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, in NIPS Deep Learning and Representation Learning Workshop, 2014.

[41] S. Hochreiter, *The vanishing gradient problem during learning recurrent neural nets and problem solutions*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6 (1998), pp. 107–116.

[42] L. Huang, S. Fayong, and Y. Guo, *Structured perceptron with inexact search*, in Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Montréal, Canada, June 2012, Association for Computational Linguistics, pp. 142–151.

[43] Y. Kim and A. M. Rush, *Sequence-level knowledge distillation*, in Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, Texas, Nov. 2016, Association for Computational Linguistics, pp. 1317–1327.

[44] A. R. Klivans and A. A. Sherstov, *Cryptographic hardness for learning intersections of halfspaces*, in 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings, 2006, pp. 553–562.

[45] D. Koller and N. Friedman, *Probabilistic Graphical Models - Principles and Techniques*, MIT Press, Cambridge, Massachusetts, 2009.

[46] N. Komodakis, N. Paragios, and G. Tziritas, *MRF optimization via dual decomposition: Message-passing revisited*, in IEEE 11th International Conference on Computer Vision, Rio de Janeiro, Brazil, 2007, pp. 1–8.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[48] G. Kundu, V. Srikumar, and D. Roth, *Margin-based decomposed amortized inference*, in Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Sofia, Bulgaria, Aug. 2013, Association for Computational Linguistics, pp. 905–913.

[49] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*, in Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, San Francisco, CA, USA, 2001, Morgan Kaufmann Publishers Inc., pp. 282–289.

[50] N. Lavrac and S. Dzeroski, *Inductive logic programming - techniques and applications*, Ellis Horwood series in artificial intelligence, Ellis Horwood, 1994.

[51] C. Lemaréchal, *Lagrangian relaxation*, in Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School, Schloß Dagstuhl, Germany, 15-19 May 2000], 2001, pp. 112–156.

[52] Q. Li, H. Ji, and L. Huang, *Joint event extraction via structured prediction with global features*, in Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Sofia, Bulgaria, Aug. 2013, Association for Computational Linguistics, pp. 73–82.

[53] R. Livni, S. Shalev-Shwartz, and O. Shamir, *On the computational efficiency of training neural networks*, in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., Curran Associates, Inc., 2014, pp. 855–863.

[54] C. Ma, J. R. Doppa, J. W. Orr, P. Mannem, X. Fern, T. Dietterich, and P. Tadepalli, *Prune-and-score: Learning for greedy coreference resolution*, in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, Oct. 2014, Association for Computational Linguistics, pp. 2115–2126.

[55] A. L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, in ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.

[56] W. Maass, G. Schnitger, and E. D. Sontag, *On the computational power of sigmoid versus boolean threshold circuits*, in 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1991, pp. 767–776.

[57] W. Maass, G. Schnitger, and E. D. Sontag, *A comparison of the computational power of sigmoid and boolean threshold circuits*, Theoretical Advances in Neural Computation and Learning, (1994), pp. 127–151.

[58] G. S. Mann and A. McCallum, *Generalized expectation criteria for semi-supervised learning of conditional random fields*, in Proceedings of ACL-08: HLT, Columbus, Ohio, June 2008, Association for Computational Linguistics, pp. 870–878.

[59] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, *The Stanford CoreNLP natural language processing toolkit*, in Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Baltimore, Maryland, June 2014, Association for Computational Linguistics, pp. 55–60.

[60] J. Martens, A. Chattopadhya, T. Pitassi, and R. Zemel, *On the representational efficiency of restricted boltzmann machines*, in Advances in Neural Information Processing Systems 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds., Curran Associates, Inc., 2013, pp. 2877–2885.

[61] A. Martins, N. Smith, and E. Xing, *Concise integer linear programming formulations for dependency parsing*, in Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language

Processing of the AFNLP, Suntec, Singapore, Aug. 2009, Association for Computational Linguistics, pp. 342–350.

[62] A. McCallum, D. Freitag, and F. C. N. Pereira, *Maximum entropy markov models for information extraction and segmentation*, in Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00, San Francisco, CA, USA, 2000, Morgan Kaufmann Publishers Inc., pp. 591–598.

[63] D. Michie, *"Memo" Functions and Machine Learning*, Nature, 218 (1968), p. 19.

[64] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, *On the number of linear regions of deep neural networks*, in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., Curran Associates, Inc., 2014, pp. 2924–2932.

[65] S. Muggleton, *Inverse entailment and progol*, New Generation Computing, 13 (1995), pp. 245–286.

[66] S. Muggleton and L. de Raedt, *Inductive logic programming: Theory and methods*, The Journal of Logic Programming, 19-20 (1994), pp. 629 – 679. Special Issue: Ten Years of Logic Programming.

[67] V. Nair and G. E. Hinton, *Rectified linear units improve restricted Boltzmann machines*, in Proceedings of the 27th International Conference on Machine Learning (ICML-10), J. Fürnkranz and T. Joachims, eds., Haifa, Israel, June 2010, Omnipress, pp. 807–814.

[68] S. Nowozin and C. H. Lampert, *Structured learning and prediction in computer vision*, Foundations and Trends in Computer Graphics and Vision, 6 (2011), pp. 185–365.

[69] D. Page and A. Srinivasan, *ILP: A Short Look Back and a Longer Look Forward*, Journal of Machine Learning Research, 4 (2003), pp. 415–430.

[70] X. Pan and V. Srikumar, *Expressiveness of rectifier networks*, in Proceedings of The 33rd International Conference on Machine Learning, M. F. Balcan and K. Q. Weinberger, eds., vol. 48 of Proceedings of Machine Learning Research, New York, New York, USA, 20–22 Jun 2016, PMLR, pp. 2427–2435.

[71] X. Pan and V. Srikumar, *Learning to speed up structured output prediction*, in Proceedings of the 35th International Conference on Machine Learning, J. Dy and A. Krause, eds., vol. 80 of Proceedings of Machine Learning Research, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018, PMLR, pp. 3996–4005.

[72] J. D. Park and A. Darwiche, *Complexity results and approximation strategies for MAP explanations*, J. Artif. Intell. Res., 21 (2004), pp. 101–133.

[73] R. Pascanu, G. Montufar, and Y. Bengio, *On the number of response regions of deep feedforward networks with piecewise linear activations*, in International Conference on Learning Representations, 2014, pp. 1–17.

[74] V. Punyakanok, D. Roth, and W.-t. Yih, *The importance of syntactic parsing and inference in semantic role labeling*, Comput. Linguist., 34 (2008), pp. 257–287.

[75] V. Punyakanok, D. Roth, W.-t. Yih, and D. Zimak, *Learning and inference over constrained output*, in Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05, San Francisco, CA, USA, 2005, Morgan Kaufmann Publishers Inc., pp. 1124–1129.

[76] J. Quinlan, *Learning logical definitions from relations*, Machine Learning, 5 (1990), pp. 239–266.

[77] J. R. Quinlan, *Induction of decision trees*, Machine Learning, 1 (1986), pp. 81–106.

[78] L. D. Raedt, A. Passerini, and S. Teso, *Learning constraints from examples*, in Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, 2018, pp. 7965–7970.

[79] L. Ramshaw and M. Marcus, *Text chunking using transformation-based learning*, in Third Workshop on Very Large Corpora, 1995.

[80] M. Richardson and P. Domingos, *Markov logic networks*, Machine Learning, 62 (2006), pp. 107–136.

[81] S. Riedel and J. Clarke, *Incremental integer linear programming for non-projective dependency parsing*, in Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, Sydney, Australia, July 2006, Association for Computational Linguistics, pp. 129–137.

[82] S. Riedel and I. Meza-Ruiz, *Collective semantic role labelling with Markov logic*, in CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning, Manchester, England, Aug. 2008, Coling 2008 Organizing Committee, pp. 193–197.

[83] S. Ross and J. A. Bagnell, *Reinforcement and imitation learning via interactive no-regret learning*, CoRR, abs/1406.5979 (2014).

[84] S. Ross, G. Gordon, and D. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, G. Gordon, D. Dunson, and M. Dudík, eds., vol. 15 of Proceedings of Machine Learning Research, Fort Lauderdale, FL, USA, 11–13 Apr 2011, PMLR, pp. 627–635.

[85] D. Roth and W.-t. Yih, *A linear programming formulation for global inference in natural language tasks*, in Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004, Boston, Massachusetts, USA, May 6 - May 7 2004, Association for Computational Linguistics, pp. 1–8.

[86] ——, *Integer linear programming inference for conditional random fields*, in Proceedings of the 22Nd International Conference on Machine Learning, ICML '05, New York, NY, USA, 2005, ACM, pp. 736–743.

[87] A. M. Rush and M. Collins, *Exact decoding of syntactic translation models through Lagrangian relaxation*, in Proceedings of the 49th Annual Meeting of the Association

for Computational Linguistics: Human Language Technologies, Portland, Oregon, USA, June 2011, Association for Computational Linguistics, pp. 72–82.

[88] A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola, *On dual decomposition and linear programming relaxations for natural language processing*, in Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, Cambridge, MA, Oct. 2010, Association for Computational Linguistics, pp. 1–11.

[89] S. J. Russell and P. Norvig, *Artificial intelligence - a modern approach, 2nd Edition*, Prentice Hall series in artificial intelligence, Prentice Hall, 2003.

[90] A. Schrijver, *Theory of linear and integer programming*, Wiley-Interscience series in discrete mathematics and optimization, Wiley, 1999.

[91] T. Serra, C. Tjandraatmadja, and S. Ramalingam, *Bounding and counting linear regions of deep neural networks*, in Proceedings of the 35th International Conference on Machine Learning, J. Dy and A. Krause, eds., vol. 80 of Proceedings of Machine Learning Research, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018, PMLR, pp. 4558–4566.

[92] F. Sha and F. Pereira, *Shallow parsing with conditional random fields*, in Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, 2003, pp. 213–220.

[93] D. Smith and J. Eisner, *Dependency parsing by belief propagation*, in Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, Honolulu, Hawaii, Oct. 2008, Association for Computational Linguistics, pp. 145–156.

[94] N. A. Smith, *Linguistic Structure Prediction*, Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers, 2011.

[95] N. A. Smith and J. Eisner, *Contrastive estimation: Training log-linear models on unlabeled data*, in Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), Ann Arbor, Michigan, June 2005, Association for Computational Linguistics, pp. 354–362.

[96] D. A. Sontag, T. Meltzer, A. Globerson, T. S. Jaakkola, and Y. Weiss, *Tightening LP relaxations for MAP using message passing*, in UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008, 2008, pp. 503–510.

[97] V. Srikumar, *An algebra for feature extraction*, in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, Canada, July 2017, Association for Computational Linguistics, pp. 1891–1900.

[98] V. Srikumar, G. Kundu, and D. Roth, *On amortizing inference cost for structured prediction*, in Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Jeju Island, Korea, July 2012, Association for Computational Linguistics, pp. 1114–1124.

[99] A. Stuhlmüller, J. Taylor, and N. Goodman, *Learning stochastic inverses*, in Advances in Neural Information Processing Systems 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds., Curran Associates, Inc., 2013, pp. 3048–3056.

[100] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds., Curran Associates, Inc., 2014, pp. 3104–3112.

[101] B. Taskar, C. Guestrin, and D. Koller, *Max-margin markov networks*, in Advances in Neural Information Processing Systems 16, S. Thrun, L. K. Saul, and B. Schölkopf, eds., MIT Press, 2004, pp. 25–32.

[102] M. Telgarsky, *Representation benefits of deep feedforward networks*, CoRR, abs/1509.08101 (2015).

[103] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, *Support vector machine learning for interdependent and structured output spaces*, in Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004, 2004.

[104] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., Curran Associates, Inc., 2017, pp. 5998–6008.

[105] T. Vieira and J. Eisner, *Learning to prune: Exploring the frontier of fast and accurate parsing*, Transactions of the Association for Computational Linguistics, 5 (2017), pp. 263–278.

[106] W. Y. Wang, K. Mazaitis, and W. W. Cohen, *Programming with personalized pagerank: A locally groundable first-order probabilistic logic*, in Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management, CIKM '13, New York, NY, USA, 2013, ACM, pp. 2129–2138.

[107] ——, *Structure learning via parameter learning*, in Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM '14, New York, NY, USA, 2014, ACM, pp. 1199–1208.

[108] W. Y. Wang, K. Mazaitis, and W. W. Cohen, *A soft version of predicate invention based on structured sparsity*, in Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, 2015, pp. 3918–3924.

[109] S. Wiseman and A. M. Rush, *Sequence-to-sequence learning as beam-search optimization*, in Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, Texas, Nov. 2016, Association for Computational Linguistics, pp. 1296–1306.

[110] Y. Xu, A. Fern, and S. W. Yoon, *Learning linear ranking functions for beam search with application to planning*, J. Mach. Learn. Res., 10 (2009), pp. 1571–1610.

[111] T. Zaslavsky, *Facing up to arrangements: face-count formulas for partitions of space by hyperplanes*, Number 154 in Memoirs of the American Mathematical Society, American Mathematical Society, Providence, RI, 1975.

[112] M. D. Zeiler, M. Ranzato, R. Monga, M. Z. Mao, K. Yang, Q. V. Le, P. Nguyen, A. W. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton, *On rectified linear units for speech processing*, in IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013, 2013, pp. 3517–3521.