Non-linear models for dependency parsing

Xudong Zhang

Université Paris XIII - Sorbonne Paris Nord
École Doctorale Sciences, Technologies, Santé Galilée

MODÈLES NON-LINÉAIRES POUR L'ANALYSE SYNTAXIQUE DES DÉPENDANCES

NON-LINEAR MODELS FOR DEPENDENCY PARSING

THÈSE DE DOCTORAT

Présentée par

XUDONG ZHANG
LIPN CNRS UMR 7030

pour l'obtention du grade de
DOCTEUR EN INFORMATIQUE

soutenue le 11 mai devant le jury d'examen composé de :

FROMONT Elisa, Université Rennes 1, Présente du jury
NASR Alexis, Université Aix-Marseille, Rapporteur
ALLAUZEN Alexandre, Université Paris Dauphine, Rapporteur
WOLFER CALVO Roberto, Université Sorbonne Paris Nord, Examinateur
CHARNOIS Thierry, Université Sorbonne Paris Nord, Directeur de thèse
LE ROUX Joseph, Université Sorbonne Paris Nord, Directeur de thèse

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to firstly express my gratitude to my supervisors Thierry Charnois and Joseph Le Roux, for their invaluable mentorship over the three years. I would like to thank their in-depth participation of the discussions, accompanied often with constrictive suggestions, which inspired the progress of the thesis. The thesis is hard, but I was lucky enough to have them as my supervisors.

Working and studying in Team RCLN@LIPN is an enjoyable and unforgettable experience. I would like to express my gratitude to the whole team. Particularly, I would like to thank Brigitte GUEVENEUX for helping with many administrative procedures, and Jorge Garcia Flores for helping in preparing the data of experiments.

I wish to extend my special thanks to my parents, who constantly supported me during the thesis. And Thanks to my friends, especially Cheng Shan, Runtian Zhang for their encouragement and drinks we shared.

My Ph.D. studies were supported by a public grant overseen by the French National Research Agency (ANR) as part of the Investissements d'Avenir program (ANR-10-LABX-0083) and IDRIS Jean-Zay (AD011011147R1)

ABSTRACT

The task of Dependency parsing aims to extract the grammatical structure of the sentence, represented with a directed graph called *parse*. A parse connects words in the sentence with arcs. Dependency Parsing can be solved with graph-based models, which evaluate the score of all parses of the sentence and use methods from graph theory (Maximum Spanning Tree for instance) to build the parses. Scores functions of graph-based models can be viewed as polynomial functions, with terms in the polynomial functions represent the score of sub-structures (combination of arcs) of the parse. Inference with graph-based models aims to find the parse which maximizes the function score. To have efficient inference algorithms, the score functions of the model are restricted to be constrained polynomial functions, which can only incorporate the score of particular sub-structures. For example, for models with first-order score functions called arc-factored models, the score function is calculated as the sum over the score of arcs. Inference for arc-factored models is efficient with Eisner or Chu-Liu-Edmonds algorithm. Higher-order polynomial functions can be used as score function for higher-order models. However, only particular combinations of arcs (sibling, grandchild) are allowed to ensure the existence of efficient inference algorithms. This gives us a constrained higher-order polynomial function in which some of the higher-order terms are acceptable. In practice, higher-order models have generally better performances than arc-factored models.

Graph-based models can be viewed as energy-based models (EBMs). An EBM takes the input data (the sentence and the parse for dependency parsing) as variables and gives a scalar called energy as output. The energy is a measure of the adequacy of the input data (the adequacy of parse to the sentence for dependency parsing). Customarily, inference with EBMs aims to minimize the energy while inference of graph-based models aims to maximize the score. The minor difference can be eliminated by taking the the negative value of the score. In this thesis, we view graph-based models as EBMs and we use learning method of EBMs to train graph-based models.

The performance of graph-based dependency parsing can be greatly improved with the use of deep neural networks, which can extract abundant information from the sentence. Arc-factored and higher-order models have been reconstructed with deep neural networks, with batchified and gradient-based inference algorithms to benefit from the acceleration of modern GPUs.

In this thesis, we focus on non-linear graph-based models with neural networks. The goal is to generalize constrained polynomial functions into more generalized forms, and to explore whether this can benefit the performance of dependency parsing, as well as the existence of efficient inference algorithms.

We firstly study a mixture-of-experts (MoE) model, with arc-factored or second order

model as experts. MoE has the potential to approximate any non-linear models with sufficient experts. We find that an averaging MoE with uniform weights of experts can improve the performance by reducing the variance of the system. We propose a stabilized training method for MoE based on Estimation-Maximization (EM) algorithm, which avoids the degeneration problem in learning MoE. We achieve new state-of-the-art with average MoE.

A generalized polynomial models is then presented, in which the score of all possible sub-structures can be incorporated into the score function. We propose efficient inference algorithms based on coordinate ascent and differentiable programming. Instead of using hinge loss for learning, we propose a new learning method based on linear approximation of higher-order models. We achieve new state-of-the-art with this method.

We then study the general non-linear models for dependency parsing, in which we assume the score function is general non-linear functions without additional constraints. For instance, non-polynomial functions can be considered as score functions, in which we have no explicit score of sub-structures, but only the score of the parse. We adapt Frank-Wolfe algorithm for training non-linear models with hinge loss. Methods of backtracking line-search, restart and early stopping are used to accelerate the convergence. We also propose a probabilistic inference network for learning probabilistic models with non-linear score functions. The original Inference network (generator) uses a GAN-like structure to maximize the non-linear score function (discriminator). In this work, we propose to use inference network to approximate the distribution of the discriminator by minimizing KL-divergence or Jensen-Shannon (JS) divergence. The calculation of loss functions are intractable for both the discriminator and the generator with non-linear models. We solve the problem by estimating directly the gradient of the loss function with sampling and MCMC methods.

FRANÇAIS

Dans cette thèse, nous nous concentrons sur les modèles non linéaires basés sur les graphes pour l'analyse syntaxique des dépendances avec des réseaux neuronaux. L'objectif est de généraliser les fonctions polynomiales contraintes en des formes plus généralisées, et d'explorer si cela peut bénéficier aux performances de l'analyse syntaxique des dépendances, ainsi qu'à l'existence d'algorithmes d'inférence efficaces.

Nous étudions tout d'abord un modèle de mélange d'experts (MoE), avec des modèles d'arc ou de second ordre comme experts. MoE a le potentiel d'approximer tout modèle non linéaire avec suffisamment d'experts. Nous constatons qu'un MoE moyenné avec des poids uniformes des experts peut améliorer les performances en réduisant la variance du système. Nous proposons une méthode d'apprentissage stabilisée pour le MoE basée sur l'algorithme d'espérance-maximisation (EM), qui évite le problème de dégénérescence dans l'apprentissage du MoE. Nous atteignons un nouvel état de l'art avec le MoE moyenné.

Nous présentons ensuite un modèle polynomial généralisé, dans lequel le score de toutes les sous-structures possibles peut être incorporé dans la fonction de score. Nous proposons des algorithmes d'inférence efficaces basés sur l'ascension par coordonnées et la programmation différentiable. Au lieu d'utiliser le 'hinge' pour l'apprentissage, nous proposons une nouvelle méthode d'apprentissage basée sur l'approximation linéaire des modèles d'ordre supérieur. Nous atteignons l'état de l'art avec cette méthode.

Nous étudions ensuite les modèles non linéaires généraux pour l'analyse syntaxique des dépendances, dans lesquels nous supposons que la fonction de score est une fonction non linéaire générale sans contraintes supplémentaires. Par exemple, les fonctions non polynomiales peuvent être considérées comme des fonctions de score, dans lesquelles nous n'avons pas de score explicite des sous-structures, mais seulement le score de l'analyse syntaxique. Nous adaptons l'algorithme de Frank-Wolfe pour l'entraînement de modèles non linéaires avec 'hinge'. Nous utilisons des méthodes de recherche linéaire avec retour en arrière, redémarrage et arrêt précoce pour accélérer la convergence. Nous proposons également un réseau d'inférence probabiliste pour l'apprentissage de modèles probabilistes avec des fonctions de score non linéaires. Le réseau d'inférence original (générateur) utilise une structure de type GAN pour maximiser la fonction de score non linéaire (discriminateur). Dans ce travail, nous proposons d'utiliser le réseau d'inférence pour approximer la distribution du discriminateur en minimisant la divergence KL ou la divergence Jensen-Shannon (JS). Le calcul des fonctions de perte est intractable à la fois pour le discriminateur et le générateur avec des modèles non linéaires. Nous résolvons le problème en estimant directement le gradient de la fonction de perte avec des méthodes d'échantillonnage et MCMC.

CHAPTER 1
INTRODUCTION

1.1 Dependency Parsing

In this thesis, we concentrate on the task of dependency parsing. The task focuses on

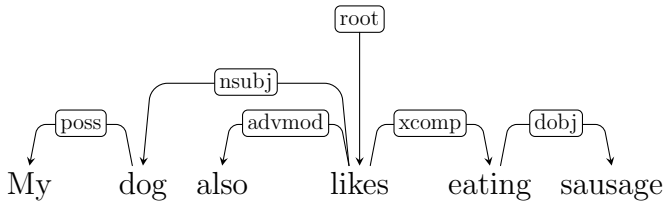

Figure 1.1: An example of Dependency Parsing

extracting the grammatical structure of a sentence by using (head, modifier) relations, as well as relation types. For example, the pair of words (likes, dog) in Figure 1.1 is a (head, modifier) relation, with the word *likes* being the head and the word *dog* being the modifier. The type of the relation is *nsubj*, which represents *nominal subject* in grammar. We mention that each word in the sentence (except the root word) has one and only one head word. The extracted relations with the root word constructs a tree like structure. Customarily, it is called arborescence, parse tree or simply parse. (head, modifier) relations in the parse are called arcs.

In this thesis, we focus on the task of extracting the relations. Once the root word and all the (head, word) relations are determined for the sentence, the decision of the relation type can be solved as a classification problem. The classification problem consists in deciding the right type of the relation among the syntactic relations and we follow previous works [Dozat and Manning, 2017, Zhang et al., 2020a, Wang and Tu, 2020] to solve this problem. In the following discussions, we refer dependency parsing as the task of extracting the relations without additional precision.

There exist two main approaches for dependency parsing: graph-based models and transition-based models. We focus on graph-based dependency parsing [Eisner, 1997] in this thesis because graph-based models directly calculate the score of parses for the sentence and give a global evaluation of the parse while transition-based dependency parsing [Zhang and Nivre, 2011] introduced briefly in Chapter 3, constructs the parse in a left to right manner with transition operations, which gives a local evaluation of the parse. Inference of graph-based model aims to find the parse which maximizes the score. Depending on the form of the score function, graph-based models can be classified as arc-factored models and

higher-order models. Arc-factored models calculate the score by making a sum over score of arcs. It is in fact a linear function (first-order model) to the variable representing the arcs. Efficient inference algorithms [Eisner, 1996, 1997, McDonald et al., 2005] exists for inference of arc-factored models. Higher-order models use higher-order polynomial score functions to the arc variables, in which the score of combination of arcs (sub-structure of the parse) are considered as higher-order terms of the polynomial function. To ensure the existence of efficient inference algorithms, only particular combination of arcs can be used [Koo and Collins, 2010], which constrain the score function to have particular higher-order terms. In general, higher-order models have better performance than arc-factored models [Koo and Collins, 2010, Zhang et al., 2020a, Wang and Tu, 2020].

Deep neural networks have greatly improved the performance of dependency parsing. [Kiperwasser and Goldberg, 2016] uses biLSTM [Hochreiter and Schmidhuber, 1997] to extract features used for score calculation from the sentence. The model achieved fairly good performance with arc-factored model. [Dozat and Manning, 2017] reformulate the calculation for score of arcs with a biaffine function. Experiments showed that biaffine has better performances than multi-layer perceptrons used in [Kiperwasser and Goldberg, 2016]. [Zhang et al., 2020a] studied a second order model with second order relation adjacent siblings. The adjacent siblings is a combination of two arcs which have the same head, and the modifiers are required to be adjacent words on the same side of the head word. They calculated the second-order score with a triaffine function and used batchified gradient-based inference algorithms to benefit from the acceleration of modern GPUs. Their results showed that higher-order models can still benefit the performance of dependency parsing with deep learning. [Wang and Tu, 2020] studied a second-order model with sibling and grandchild scores. The grandchild model is the combination of arcs with the modifier of one arc being the head of another arc (see Figure 3.8). Exact inference algorithms do not exist in this case so they used mean field variational inference (MFVI) to estimate the distribution of arcs. Their experiments showed that higher-order deep model can benefit the performance even with approximate inference.

In this thesis, we follow works in deep higher-order graph-based models. To ensure the existence of efficient inference algorithms, we propose a Mixture-of-Experts (MoE) [Jacobs et al., 1991, Brown and Hinton, 2001] method to approximate non-linear models with arc-factored or second order models. Then we studied a *generalized* polynomial model for graph-based dependency parsing. The score function is still polynomial, but without restriction of using particular higher-order terms. Finally, we studied FW algorithm and probabilistic inference network for graph-based model with general non-linear score functions.

1.2 Graph-based Dependency Parsing as Energy-based Models

Graph-based models calculates the score of parses for the sentence, which can be viewed as an energy-based model (EBMs) [LeCun et al., 2006]. EBMs can calculate the energy for pairs of inputs. The energy is used as a measure of adequacy of the input data. For dependency parsing with sentence and its parse as inputs, it is explained as the adequacy of using the parse as the grammatical structure of the sentence. Customarily, inference with EBMs aims to minimize the energy with part of inputs that can vary (for dependency parsing, the sentence is fixed, the parse can vary). The same thing can be done with graph-based model by taking the negative value of score.

By viewing graph-based models as EBMs, methods for learning in EBMs like structured hinge loss [Tsochantaridis et al., 2004, 2005, Taskar et al., 2003, 2005] can be used to train the model, which is well used in parser of [Kiperwasser and Goldberg, 2016]. Besides, it is possible to train EBMs as a probabilistic model by using the Boltzmann distribution [Landau and Lifshitz, 1968], i.e. assuming the probability of inputs is proportional to its exponential value of negative energy. For graph-based dependency parsing, it is equivalent to say the probability of parse is proportional to its exponential value of score. Probabilistic Models for dependency parsing under the framework of EBMs have been studied in [Eisner, 1997, Dozat and Manning, 2017, Zhang et al., 2020a, Wang and Tu, 2020] for arc-factoed and second-order models.

In this thesis, we follow previous works to work with the framework of EBMs. The energy functions (or score function) are limited as constrained polynomial functions. We propose to use a concrete framework of EBMs: Structure Prediction Energy Network (SPEN) [Belanger and McCallum, 2016] to construct generalized polynomial functions, or even general non-linear functions. SPEN calculates the energy function with two parts: the local energy and the global energy. The local energy is designed to be a linear function to part of the inputs that varies while the global energy can be general non-linear functions. In this thesis, we consider the case when the global part is a generalized polynomial function, and we also discuss the case when we have general non-linear functions for the global energy.

We mention particularly that the mixture-of-experts (MoE) method presented in Chapter 4 is not strictly under the framework of EBMs, but each expert is still an independent EBM.

1.3 Organization and Contributions

The thesis is organized as follows:

Chapter 2 and 3 are background of the thesis

- In Chapter 2, we introduce EBMs for structure prediction. We make comparison between EBMs and Feed-Forward Models (in Section 2.2) to clarify the advantages and difficulties of using EBMs. We discuss learning for EBMs with max-margin loss and negative log-likelihood. A concrete framework SPEN is presented in the end, which is used as the base for constructing our own models.

- In Chapter 3, we introduce the basis of dependency parsing. We present the history of graph-based dependency parsing, which started with the simple arc-factored models to the more advanced higher-order models. We give formally the definitions of two types of parse trees: projective trees and non-projective trees. The arc-factored model and the second order models are introduced, with a detailed introduction of inference algorithms for different models. The use of deep learning methods for dependency parsing is introduced in the end, with a detailed introduction of the architecture of deep graph-based models and batchified gradient-based inference algorithms. We introduce in the end an end-to-end method for second-order model, which inspires our work in Chapter 5.

Chapters 4, 5, 6 are contributions of the thesis.

- In Chapter 4, we introduce our MoE method for constructing non-linear models [Zhang et al., 2021]. We propose efficient inference for MoE based on Minimum Bayes-Risk (MBR) [Smith and Smith, 2007a]. We show that the average MoE with uniform weights of experts can reduce the variance of the system, which benefits the performance of dependency parsing and gives new SOTA. We stabilize the training of MoE with EM based methods which avoids the degeneration problem in training of MoE.

- In Chapter 5, we generalize the constrained higher-order polynomial models to generalized polynomial models. We show that the generalized model is capable to represent the score of all possible structures, of any order for the parse tree. We propose efficient inference algorithm based on coordinate ascent [Bertsekas, 1999] and combine it with genetic algorithms [Fukunaga, 1998] to achieve better solutions. A new learning method based on linear approximation of the score function is proposed, which has in general better performance and faster convergence than hinge loss. We achieve new SOTA with our method.

- In Chapter 6, we study the graph-based dependency parsing with general non-linear functions. We adapt FW algortihm for learning and inference. We use backtracking line-search, restart and early stopping strategies to accelerate the convergence of FW to a local optimum. FW algorithm is most adapted for hinge loss. For probabilistic

model, we propose to use the probabilistic inference network to approximate directly the distribution of the non-linear model. We propose calculable new loss functions to estimate directly the gradient of intractable loss functions. The new loss functions are calculated with sampling and we propose to use MCMC methods reduce the error of sampling.

In Chapter 7, we summarize the thesis and give future research directions.
Our contributions can be summarized as:

- We constructed a non-linear model with the mixture-of-experts (MoE), which has the potential to approximate any non-linear model with sufficient experts. We studied the simple averaging with uniform weights for experts and found that average MoE can reduce the variance of the system, which helps to increase the performance of parsing. We stabilize the training of MoE with deep graph-based experts.

- We generalized the constrained higher-order models to general polynomial models and proposed an efficient inference algorithm based on coordinate ascent. As the polynomial function is non-convex, we combined the method with genetic algorithm to produce better solution. Instead of using hinge loss for training, we proposed a new learning method with linear approximation of the score function, which has in general faster convergence and better performances

- We study the learning and inference with general non-linear score functions. We adapted the FW algorithm for dependency parsing and proposed to use backtracking line-search, restart and early stopping to accelerate the convergence. FW algorithm is used for learning with hinge loss. For learning with the probabilistic model, we proposed the probabilistic inference network. Calculable loss functions with sampling and MCMC methods are proposed based on KL-divergence or JS-divergence for learning with probabilistic inference network.

CHAPTER 2
ENERGY-BASED MODELS

2.1 Introduction

In this chapter, we introduce the Energy Based Models (EBMs) [LeCun et al., 2006, Ranzato et al., 2007, Belanger and McCallum, 2016], a framework for deep learning models. EBMs is more flexible for constructing complex models for structure predictions, which may be able to exploit more abundant information of complex structures (discussed in details in Section 2.2). We focus on graph-based dependency parsing in this work, which uses basically EBMs to construct different models. Before we introduce dependency parsing (in chapter 3), an introduction of background over EBMs is necessary and beneficial for the following discussion.

An EBM maps points in data space to a scalar which we call energy. A well trained EBM is assumed to have low energy for correct points but high energy for wrong points. EBM have been used for structure prediction [Gygli et al., 2017, Rooshenas et al., 2019, Tu and Gimpel, 2018], text generation [Deng et al., 2020], image generation [Zhao et al., 2017, Du and Mordatch, 2019, Nijkamp et al., 2020] by using different data space. For graph based dependency parsing, we use EBMs on structure prediction. The following introduction of EBMs assumes a data space for structure prediction.

In section 2.2, we introduce EBMs over structure prediction and its connections and differences to normal feed-forward models. In section 2.3, learning of EBMs is introduced, with a detailed discussion of different loss functions for probabilistic learning and max-margin learning. In section 2.4, we introduce a concrete EBM framework: Structure Prediction Energy Network (SPEN) [Belanger and McCallum, 2016] and different methods adapted for learning over the framework. For dependency parsing problem, we basically construct our model over SPEN.

The chapter is a modern introduction to EBMs, with [LeCun et al., 2006] as the main reference of the chapter. We recommend reading of the original paper for more details.

2.2 EBMs for Structure Prediction

For structure prediction, the data space is $\mathcal{X} \times \mathcal{Y}$. $\mathcal{X}$ is the set of all possible inputs while $\mathcal{Y}$ is the set of all possible outputs. For particular $x \in \mathcal{X}$, $\mathcal{Y}_x$ denote the set of all possible outputs for x. When x is clear from the context, we abuse notation and simplify $\mathcal{Y}_x$ with $\mathcal{Y}$.

For an EBM, the energy function maps the data space to a scalar, i.e.

$$E(x, y; \Theta) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \tag{2.1}$$

where Θ is the parameter of the model.

With an EBM, inference is to find the output which minimizes the energy:

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmin}} E(x, y; \Theta) \tag{2.2}$$

Inference can be hard for EBM because the energy function can be complex (non-linear, non-convex) to the variable y. Even for linear functions to y, as $\mathcal{Y}$ may be a discrete structure space, exact prediction is not trivial because $|\mathcal{Y}|$ may be large and the structure space has complex constraints.

For a subset $X \times Y \subseteq \mathcal{X} \times \mathcal{Y}$ which represents the training data, a well trained EBM is assumed to have low energy $\forall (x, y) \in X \times Y$. And $\forall (x, y) \in X \times Y$, it should give high energy for all the other wrong pairs (x, y'), with $y' \in \mathcal{Y}_x, y' \neq y$. It is not trivial to construct directly a loss function to achieve the goal and learning of EBMs are discussed in details in section 2.3.

Before we continue the discussions on EBMs, we introduce Feed-forward neural network, which is used a lot in machine learning. We call the model constructed with it as the feed-forward models (FFMs). We make comparisons between EBMs and FFMs to clarify their connections and differences. This should show the advantages and difficulties of using EBMs.

Feed-Forward Neural Network and Feed-Forward Model Feed-forward neural network [Svozil et al., 1997] is the simplest type of artificial neural networks. With the input vector x, information moves in one direction (forward), with the output vector of the neural network $\hat{y}$ represents usually the prediction. The vector of hidden layers ($h^{(t)}$) can be calculated with Multi-layer perceptrons (MLPs), which has the following form:

$$h^{(t+1)} = f_a(W^{(t)} h^{(t)} + b^{(t)})$$

with $W^{(t)} \in \mathbb{R}^{d_{out} \times d_{in}}, b^{(t)} \in \mathbb{R}^{d_{out}}$ the learnable parameters, $h^{(t)} \in \mathbb{R}^{d_{in}}$ the input vector and $h^{(t+1)} \in \mathbb{R}^{d_{out}}$ the output vector. f_a is the activation function applied over each element of the vector, which can be for example the sigmoid function : $f(h_i) = \frac{1}{1+e^{-h_i}}$. The universal approximation theorem for neural networks [Hornik et al., 1989] argues that every continuous function which maps an input interval of real numbers (x) to an output interval of real numbers ($\hat{y}$) can be approximated arbitrarily closely with MLPs of one hidden layer with enough nodes of hidden layers. Thus, we can well use the feed-forward neural network for

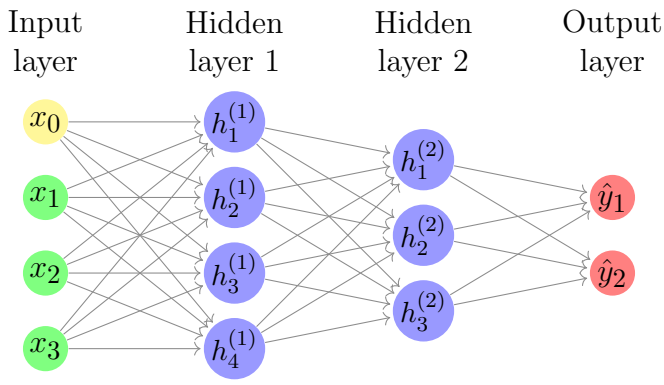

Figure 2.1: Feed-Forward Neural Network Computation

the structure prediction because the set of input $\mathcal{X}$ and the set of output $\mathcal{Y}$ can be limited to some intervals.

Unlike EBMs which map $\mathcal{X} \times \mathcal{Y}$ to a scalar $\mathbb{R}$, a FFM constructed with feed-forward neural networks maps directly the input $\mathcal{X}$ to the output $\mathcal{Y}$:

$$f(x; \Theta) : \mathcal{X} \to \mathcal{Y}$$

where Θ is the parameter of the model.

In this case, inference is direct, with

$$\hat{y} = f(x; \Theta)$$

In machine learning, the problem of: finding the parameters of the model Θ which minimize the error, is expressed as minimizing the loss function, with the loss function and the problem as:

$$L(\Theta) = \sum_{x,y \in X \times Y} D(f(x; \Theta), y)$$
$$\Theta = \underset{\Theta'}{\operatorname{argmin}}\, L(\Theta')$$

where $X \times Y$ is the training data, D measures the difference between predicted output and true output, which could be for instance $L1, L2$ norm, cross entropy.

In deep learning, gradient based optimization methods like Stochastic Gradient Descent (SGD) [Robbins and Monro, 1951, Kiefer and Wolfowitz, 1952] or Adam [Kingma and Ba, 2015] can be used to minimize the loss function to the parameters Θ, which are expected to

give:

$$\begin{aligned}\Theta &= \underset{\Theta'}{\operatorname{argmin}} \sum_{x,y \in X \times Y} D(\hat{y}, y) \\ &= \underset{\Theta'}{\operatorname{argmin}} \sum_{x,y \in X \times Y} D(f(x;\Theta'), y)\end{aligned}$$

for FFMs.

Thus for FFMs, the construction of loss function is also direct. For most problems, several measures can be used and the choice of proper measure is usually experimentally based, with the one which gives the best performance.

Connections between EBMs and FFMs If we note $E(x, y; \Theta) = D(f(x; \Theta), y)$, a FFM can be viewed directly as an EBM. Conversely, if we can have analytical solutions to $y(x; \Theta) = \operatorname{argmin}_{y' \in \mathcal{Y}} E(x, y'; \Theta)$, an EBM can be viewed as a FFM by setting $f(x; \Theta) = y(x; \Theta)$. However, this is not easy because inference of EBM is hard. For complex energy functions to the variable y, we usually have at most an approximate estimation. Even when analytical solution is possible, the argmin operation is not differentiable, which may pose problem for training in back-propagation. In this case, EBMs can be viewed as a general form of FFMs.

Difficulties and Advantages of Using EBMs In comparison with FFMs, the difficulties of using EBMs lie in both learning and inference. Unlike FFMs where the inference is direct, inference for EBMs can be intractable for that the set $\mathcal{Y}_x$ can be large and the energy function can be complex to the variable y. Due to the simplicity of inference, the construction of loss function for FFMs is also direct while the choice of proper measure can be solved by using the one which gives the best performance. For EBMs, learning could be hard because we need to have low energy for correct pairs, while high energy for all the other wrong pairs. Particular loss functions can be used to achieve the goal, which is discussed in detail in section 2.3.

The advantage of EBMs is due to their generality in form. For FFMs, although there usually exists several choices of measure, the form of energy function, if we express $E(x, y; \Theta) = D(f(x; \Theta), y)$, is highly restricted. Note that except the outer measure function, the variables x and y are fully separated. The restriction of the form may limit the model from exploring complex structures of output y. For EBMs, instead of using a measure D, the energy function $E(x, y; \Theta)$ which measures the energy to combine x to y, becomes a learnable measure function. Note that we do not restrict x and y to be separated. Thus, the information in y can be fully exploited with complex models for the energy function.

2.3 Learning with EBM

We restrict our discussion to deep learning models, i.e. models trained with loss functions and updated with gradient based optimization methods (SGD [Robbins and Monro, 1951, Kiefer and Wolfowitz, 1952], Adam [Kingma and Ba, 2015], etc). Thus, the crucial problem is how to use a proper loss function to train the EBM.

Two typical types of loss functions can be used for training EBMs (see others in [LeCun et al., 2006]): max-margin loss and negative log-likelihood.

2.3.1 Max-Margin Loss

To have lowest energy for correct outputs, we would like to have positive gap ($E(x, y; \Theta) - E(x, y'\Theta), y' \neq y$) between correct and wrong outputs. The gap is called *margin* between y and y'. Max-margin loss trains EBMs to maximize the margin. To achieve the goal, three types of loss functions can be used for max-margin loss: Energy Loss, Perception Loss and Hinge Loss.

Energy Loss Energy loss trains EBMs by minimizing directing the energy of correct outputs:

$$L_{\mathrm{E}}(\Theta) = \sum_{x,y \in X \times Y} E(x, y; \Theta) \tag{2.3}$$

The advantage is that the loss function can be easily calculated. The limitation is that decreasing the energy of correct output may not maximize the margin because the energy of wrong outputs may also decrease. Energy loss works for maximizing the margin when decreasing energy of correct outputs will automatically increase energy of wrong outputs. For example, if viewing a FFM as an EBM: $E(x, y; \Theta) = D(f(x; \Theta), y)$, the loss function of FFM is exactly the energy loss, which satisfies the condition that decreasing the energy of correct outputs increases automatically the energy of wrong outputs.

Perception Loss Unlike energy loss which only modifies the energy of correct outputs, perceptron loss train EBMs by decreasing the energy of correct outputs while increasing the energy of the most violating prediction. The most violating prediction is defined as the output which minimizes the energy, but not equal to the correct output. For a pair of data (x, y), the most violating prediction can be calculated as:

$$\hat{y}(x) = \underset{\substack{y' \in \mathcal{Y} \\ y' \neq y}}{\operatorname{argmin}} E(x, y'; \Theta)$$

The perceptron is:

$$\begin{aligned} L_{\mathrm{P}}(\Theta) &= \sum_{x,y \in X \times Y} E(x, y; \Theta) - E(x, \hat{y}(x); \Theta) \\ &= \sum_{(x,y) \in X \times Y} E(x, y; \Theta) - \min_{\substack{y' \in \mathcal{Y} \\ y' \neq y}} E(x, y'; \Theta) \end{aligned} \tag{2.4}$$

The idea of perceptron loss is that instead of maximizing margin between correct output and all the other wrong outputs, it is sufficient to maximize the margin between correct output and the most violating output. Thus even for large $\mathcal{Y}$, only two points are concerned for the calculation of loss function.

When the most violating prediction can be efficiently found, perceptron loss can well be used for training EBMs. This usually requires the energy function to have simple forms because firstly for complex energy functions, calculating the most violating prediction can be hard or intractable, which may limit the model from maximizing margins between correct and wrong outputs. Secondly, as we require $y' \neq y$. When inference of EBM gives correct output, finding the most violating prediction becomes finding the output with the second lowest energy, which may be even harder or more complex than inference.

Hinge Loss Hinge loss [Tsochantaridis et al., 2004, 2005, Taskar et al., 2003, 2005] defines the most violating prediction in a different way:

$$\hat{y}(x) = \operatorname*{argmax}_{y' \in \mathcal{Y}} \Delta(y, y') - E(x, y'; \Theta)$$

Where $\Delta(y, y')$ is a measure of distance between correct and wrong outputs with $\Delta(y, y') \geq 0, \forall y' \in \mathcal{Y}$ and $\Delta(y, y') = 0$ if and only if $y = y'$. Unlike perceptron loss which requires $y' \neq y$, the calculation of the most violating prediction in hinge loss omit this requirement by adding a penalized margin Δ. In practice, Δ is usually designed to have simple forms (for instance, a linear function to the variable y') in order to simplify the calculation. Thus the second lowest problem can be avoided.

The form of hinge loss is:

$$L_{\mathrm{H}}(\Theta) = \sum_{(x,y) \in X \times Y} [E(x, y; \Theta) + \max_{y' \in \mathcal{Y}} (\Delta(y, y') - E(x, y'; \Theta))]_+ \tag{2.5}$$

Where $[\dots]_+$ represents $\max(0, \dots)$ and is not necessary if inference is exact for the inner max[1].

1. see Appendix A.1

For hinge loss, training terminates when $E(x, y; \Theta) + \Delta(y, y') - E(x, y'; \Theta) \leq 0, \forall y' \in \mathcal{Y}$, i.e.

$$E(x, y'; \theta) - E(x, y; \Theta) \geq \Delta(y, y'), \forall y' \in \mathcal{Y}$$

Thus, training terminates if and only if the margin between correct and wrong outputs is at least $\Delta(y, y')$. Unlike perceptron loss which continues to increase the margin to $+\infty$, hinge loss terminates once the penalized margin is reached.

2.3.2 Negative Log-likelihood

EBMs can be transformed to probabilistic models by assuming $p(y|x; \Theta) \propto -E(x, y; \Theta)$. Thus, instead of creating margin between correct and wrong outputs, it is sufficient to maximize the probability of correct output by using negative log-likelihood:

$$\begin{aligned} L_{\text{NLL}}(\Theta) &= \sum_{(x,y) \in X \times Y} -\log p(y|x; \Theta) \\ &= \sum_{(x,y) \in X \times Y} -\log \frac{\exp(-E(x, y; \Theta))}{\sum_{y' \in \mathcal{Y}} \exp(-E(x, y'; \Theta))} \\ &= \sum_{(x,y) \in X \times Y} E(x, y; \Theta) + \log \sum_{y' \in \mathcal{Y}} \exp(-E(x, y'; \Theta)) \\ &= \sum_{(x,y) \in X \times Y} E(x, y; \Theta) + \log Z(x; \Theta) \end{aligned} \tag{2.6}$$

where $Z(x; \Theta) = \sum_{y' \in \mathcal{Y}} \exp(-E(x, y'; \Theta))$ is called the partition function.

The calculation of $L_{\text{NLL}}(\Theta)$ requires the calculation of the partition function, which may be intractable for large set $\mathcal{Y}$ or for complex energy functions. For dependency parsing, where $|\mathcal{Y}|$ is exponential to the length of the sentence, it is impossible to calculate the energy for all possible arborescences. Thus, constraints are added over the form of the score function and the structure to ensure the existence of efficient algorithms [Eisner, 1996, 1997, Koo and Collins, 2010, Koo et al., 2010].

When the partition function cannot be calculated directly, methods based on sampling can be used instead to estimate the gradient. We calculate the gradient of $L_{\text{NLL}}(\Theta)$:

$$\frac{dL_{\text{NLL}}(\Theta)}{d\Theta} = \sum_{(x,y) \in X \times Y} \frac{\partial E(x, y; \Theta)}{\partial \Theta} + \frac{\partial \log Z(x; \Theta)}{\partial \Theta}$$

The second term can be expanded as:

$$\begin{aligned}\frac{\partial \log Z(x;\Theta)}{\partial \Theta} &= \frac{1}{Z(x;\Theta)}\frac{\partial Z(x;\Theta)}{\partial \Theta} \\ &= \frac{1}{Z(x;\Theta)}\frac{\sum_{y'\in\mathcal{Y}}\exp(-E(x,y';\Theta))}{\partial \Theta} \\ &= -\sum_{y'\in\mathcal{Y}}\frac{\exp(-E(x,y';\Theta))}{Z(x;\Theta)}\frac{\partial E(x,y';\Theta)}{\partial \Theta} \\ &= -\mathbb{E}_{y'\sim p(y|x;\Theta)}\frac{\partial E(x,y';\Theta)}{\partial \Theta}\end{aligned}$$

where $\mathbb{E}_{y'\sim p(y|x;\Theta)}\frac{\partial E(x,y';\Theta)}{\partial \Theta}$ represents the expectation of $\frac{\partial E(x,y';\Theta)}{\partial \Theta}$, with $p(y|x;\Theta)$ the distribution of the random variable y'.

Thus the gradient of negative log-likelihood can be written as:

$$\begin{aligned}\frac{dL_{\text{NLL}}(\Theta)}{d\Theta} &= \sum_{(x,y)\in X\times Y}\frac{\partial E(x,y;\Theta)}{\partial \Theta} - \mathbb{E}_{y'\sim p(y|x;\Theta)}\frac{\partial E(x,y';\Theta)}{\partial \Theta} \\ &= \sum_{(x,y)\in X\times Y}\frac{\partial}{\partial \Theta}[E(x,y;\Theta) - \mathbb{E}_{y'\sim p(y|x;\Theta)}E(x,y';\Theta)]\end{aligned} \tag{2.7}$$

By using the gradient to minimize the loss function, we decrease the energy of *correct* while increase the expectation of energy. When calculation of the partition function is impossible, equation (2.7) implies that the gradient can be estimated by sampling K samples from the distribution $p(y|x;\Theta)$ according to the law of large numbers [Grinstead and Snell, 1997].

$$\frac{dL_{\text{NLL}}(\Theta)}{d\Theta} = \sum_{(x,y)\in X\times Y}\frac{\partial}{\partial \Theta}[E(x,y;\Theta) - \frac{1}{K}\sum_{k=1}^{K}E(x,y^{(k)};\Theta)]$$

with $\{y^{(1)},\ldots,y^{(K)}\} \sim p(y|x;\Theta)$.

When sampling from exact distribution is hard, approximate methods can be applied. Here we adopt two methods for sampling.

Importance Sampling [Kloek and Van Dijk, 1978] We use a simple-to-sample distri-

bution $Q(y)$. The deduction of importance sampling is shown below:

$$\begin{aligned}
\mathbb{E}_{y' \sim p(y|x;\Theta)} E(x, y'; \Theta) &= \mathbb{E}_{y' \sim Q(y)} \frac{p(y'|x;\Theta)}{Q(y')} E(x, y'; \Theta) \\
&= \mathbb{E}_{y' \sim Q(y)} \frac{\exp(-E(x, y'; \theta))}{Q(y')} \frac{1}{\sum_{y'' \in \mathcal{Y}} \exp(-E(x, y''; \Theta))} E(x, y'; \Theta) \\
&= \mathbb{E}_{y' \sim Q(y)} \frac{\frac{\exp(-E(x,y';\theta))}{Q(y')}}{\mathbb{E}_{y'' \sim Q(y)} \frac{\exp(-E(x,y'';\theta))}{Q(y'')}} E(x, y'; \Theta) \\
&= \mathbb{E}_{y' \sim Q(y)} \omega(y') E(x, y'; \Theta)
\end{aligned}$$

With the last equation, we apply sampling from the distribution $Q(y)$ and correct the expectation of energy with weight $\omega(y')$. Importance sampling can make learning more stable. This is because even the distribution $Q(y)$ differs a lot to the true distribution $p(y|x;\Theta)$. the weight $\omega(y')$ can give higher weights for samples with lower energy while smaller weights for samples with high energy. This can compensate to some extend the error of sampling.

Contrastive Divergence [Hinton, 2002] We use a distribution $Q(y)$ where sampling is easy. Unlike importance sampling, we use MCMC [Grimmett and Stirzaker, 2020, Andrieu et al., 2003] methods to approximate sampling from $p(y|x;\Theta)$. [Hinton, 2002] shows that the method can give samples with low variance for problems which fit Gibbs Sampling even with few number of steps.

2.3.3 Connections Between Max-Margin Loss and Negative Log-Likelihood

Negative Log-Likelihood As Max-Margin Loss Negative Log-Likelihood creates implicitly an infinite margin between correct prediction and wrong predictions. This can be seen from the terminal condition of training. For Negative Log-Likelihood, training terminates when $L_{\text{NLL}}(\Theta) \to 0$, which is equivalent to $\frac{\exp(-E(x,y;\Theta))}{Z(x;\Theta)} \to 1$. The equation can be

developed as:

$$
\begin{aligned}
\frac{\exp(-E(x,y;\Theta))}{Z(x;\Theta)} \to 1 &\iff \frac{\exp(-E(x,y;\Theta))}{\sum_{y'\in\mathcal{Y}}\exp(-E(x,y';\Theta))} \to 1 \\
&\iff \frac{1}{1+\sum_{\substack{y'\in\mathcal{Y}\\ y'\neq y}}\exp(E(x,y;\Theta)-\exp(E(x,y';\Theta)))} \to 1 \\
&\iff \sum_{\substack{y'\in\mathcal{Y}\\ y'\neq y}}\exp(E(x,y;\Theta)-\exp(E(x,y';\Theta))) \to 0 \\
&\iff E(x,y';\Theta)-E(x,y;\Theta) \to +\infty, \forall y'\in\mathcal{Y}, y'\neq y
\end{aligned}
\tag{2.8}
$$

The last equation indicates that the margin between the correct output and any wrong output should approach infinity. Thus, with negative log-likelihood, we require implicitly an infinite margin between correct and wrong outputs.

When comparing equation (2.7) with the perceptron loss (equation (2.4)), we can find that the forms of equation are quite similar. The difference is that for perceptron loss, the most violating prediction is calculated with the operation max while for negative log-likelihood, the most violating prediction is calculated with softmax, with the probability of prediction proportional to the exponential value of negative energy. Note that when applying one sample estimation for equation (2.7), the most violating prediction in perceptron loss can be viewed as a good sample for that it has the second lowest energy in the worst case, thus there is a high probability (the second highest in the worst case) that the most violating prediction in perceptron loss will be chosen.

Inspired by equation (2.7), samplerank [Wick et al., 2011, Gao and Gormley, 2020] can be used for training EBMs. The method can be described briefly as using MCMC methods to sample *wrong* predictions while the loss is calculated as the sum of hinge loss with sampled *wrong* predictions. The method is shown to achieve better performances than contrastive divergence [Wick et al., 2011]

Max-Margin Loss as Negative Log-Likelihood Perceptron loss can be viewed as a rigid version of negative log-likelihood because it uses max instead of softmax for the calculation of the most violating prediction. Besides, it also requires an infinite margin. For perceptron loss, we have $L_{\mathrm{P}} \to -\infty$ at the end of training. It is equivalent to $E(x,y';\Theta) - E(x,y;\Theta) \to +\infty, \forall y' \in \mathcal{Y}, y' \neq y$ which is same to equation (2.8). Thus for perceptron loss, it trains the model to make the probability of correct outputs to 1.

For hinge loss, we also use rigid calculation of the most violating prediction. However, we require only a limit margin between correct and wrong outputs. Training with hinge loss

terminates when $E(x, y'; \theta) - E(x, y; \Theta) \geq \Delta(y, y'), \forall y' \in \mathcal{Y}$, which is equivalent as:

$$\frac{p(y|x; \Theta)}{p(y'|x; \Theta)} \geq \exp(\Delta(y, y')), \forall y' \in \mathcal{Y}$$

Thus, hinge loss augments the probability of correct output y until it is at least $\exp(\Delta(y, y'))$ times larger than all possible $y' \in \mathcal{Y}$.

2.4 A Concrete Framework: Structure Prediction Energy Networks (SPEN)

SPEN [Belanger and McCallum, 2016] is a flexible framework for structure prediction based on EBMs. The crucial idea of SPEN is that the energy of network is separated into two parts: local energy and global energy:

$$E(x, y; \Theta) = E_l(x, y; \Theta_l) + E_g(x, y; \Theta_g) \tag{2.9}$$

The local energy $E_l(x, y; \Theta_l)$ is chosen to be a linear function to the variable y, which aims to capture the direct relations of the input data (x, y). The global energy $E_g(x, y; \Theta_g)$ can be complex non-linear functions to the variable y, which aims to capture complex structures in y. A typical structure of SPEN is shown below: where $F(x)$ is the feature vectors extracted

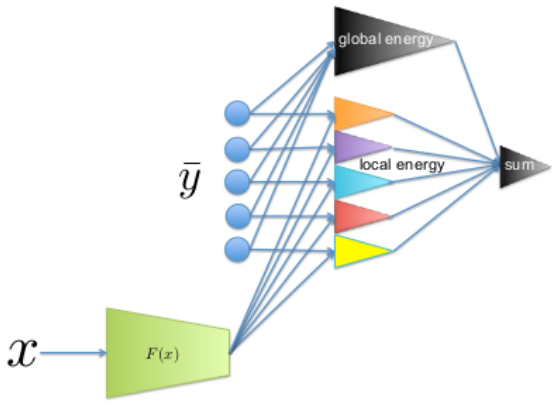

Figure 2.2: SPEN Architecture for Multi-Label Classification

from input x. The local energy is a sum of scores for labels while the global energy can be calculated with a complex (non-linear) black box.

2.4.1 Learning of SPEN with Hinge Loss

In order to calculate the hinge loss, the most *violating* prediction needed to be calculated in advance. In [Belanger and McCallum, 2016], they do not assume a special form of the global energy. When the set size $|\mathcal{Y}|$ is not large, energy of all $y \in \mathcal{Y}$ can be calculated to selected the most violating prediction. If it is not the case, gradient based methods can be used. We adopt to present three methods for solving the problem, with the first one which can guarantees the convergence to the global optimum, while the other two can guarantee the convergence to a local optimum.

Input Convex Neural Network [Amos et al., 2017] In order to guarantee the convergence to the global optimum with gradient based methods, the energy function needs to be convex to the variable y, which can be realized with Input Convex Neural Networks. The construction is based on two properties of convex functions [Boyd et al., 2004]:

1. If f, g two convex functions, then $af + bg$ is also convex $\forall a, b \in \mathbb{R}, a, b \geq 0$.
2. If g is convex and f is convex and non-decreasing, then $f(g)$ is also convex.

For MLPs used in deep learning with form:

$$\text{MLP}(x) = f_a(Wx + b)$$

with $x \in \mathbb{R}^{d_{\text{in}}}$ the input vector, $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, b \in \mathbb{R}^{d_{\text{out}}}$ the parameters of MLP and f_a the non-linear activation function applying over every element of the vector.

To satisfy the first condition, the parameters W are set to be non-negative by taking its absolute value $|W|$. To satisfy the second condition, the activation function is restricted to be convex non-decreasing. Thus activation functions like ReLU, ELU [Sharma et al., 2017] can be used.

Thus, starting with a linear function (multi-layer perceptron (MLP)), which is convex, we can construct non-linear convex functions by using non-negative parameters in MLPs and convex non-decreasing activation functions (ReLU, ELU, etc). For discrete or non-convex set $\mathcal{Y}$, it is sufficient to expand $\mathcal{Y}$ to a convex compact set with convex combination. Thus convex optimization methods could be used and we can guarantee the convergence to the global optimum. [Amos et al., 2017] show that the method can obtain state-of-the-art results on tasks of multi-label classification, image completion and continuous action reinforcement learning, but [Belanger et al., 2017] reported that forcing the global energy to be convex made performances worse on tasks of image denoising and semantic role labeling. They explain the reason as: only MLPs with non-negative parameters and convex non-decreasing activation functions can be used, which may limit the capacity of the neural network.

End to End Learning [Belanger et al., 2017] For simple gradient descent with fixed

step T, the most violaating predictions can be approximated with:

$$\hat{y} \approx y_T = y_0 - \sum_{t=0}^{T-1} \eta_t \frac{\partial E(x, y_t; \Theta) + \Delta(y, y_t)}{\partial y_t}$$

where η_t is the step-size, $\hat{y}$ the most violating prediction. Here we use y_T to approximate the most violating prediction.

Back-propagation will introduce the second order terms of gradient by using y_T for the calculation of the hinge loss. This can be solved by unrolling the optimization with finite difference matrix-vector products [Domke, 2012].

Inference Network Instead of using optimization methods to do inference directly, inference network Tu and Gimpel [2018] propose to use a neural network for optimization. The additional neural network (Generator) has output $y = G(x; \Phi)$, with Φ the parameters of the neural network. The new network is trained to find the most *violating* prediction, i.e.

$$L_{\mathbf{G}}(\Phi) = \sum_{(x,y) \in X \times Y} E(x, G(x; \Phi); \Theta) - \Delta(y, G(x; \Phi))$$

And the output of network G is used in hinge loss:

$$L_{\mathrm{H}}(\Theta) = \sum_{(x,y) \in X \times Y} \max(0, E(x, y; \Theta) + \Delta(y, G(x; \Phi)) - E(x, G(x; \Phi); \Theta))$$

This leads to in fact a saddle point optimizing problem:

$$\min_{\Theta} \max_{\Phi} \sum_{(x,y) \in X \times Y} \max(0, E(x, y; \Theta) + \Delta(y, G(x; \Phi)) - E(x, G(x; \Phi); \Theta)) \tag{2.10}$$

We could find that the method is similar to GAN [Goodfellow et al., 2014] and techniques like regularization and pre-training are required to ensure stability of learning.

2.5 Conclusion

In this chapter, we introduce the background of EBMs for structure prediction.

We present the FFMs and EBMs for comparison. We present firstly that learning and inference with FFMs are direct and simple, but may require additional efforts for EBMs. We show that EBMs can be viewed as a generality of FFMs. The generality in form brings difficulty for learning and inference, but also possibility for exploiting information of complex structures.

We present two types of loss functions for learning EBMs: max-margin loss and negative log-likelihood. We discuss in general the calculation of the loss functions. For max-margin loss, a maximization sub-problem needed to be solved (except energy loss), which can be viewed as an sub-inference problem for calculating the loss function. For negative log-likelihood, the partition function needed to be calculated. We also show that instead of calculating directly the negative log-likelihood, we can estimate directly the gradient of the loss function with sampling methods. Comparisons are made between these two types of loss functions, in which we find that negative log-likelihood can be viewed as a soft version (most violating prediction calculated with softmax) of perceptron loss (most violating prediction calculated with max). While both perceptron loss and negative log-likelihood require an infinite margin or the probability of correct output equals to one, hinge loss has a less strict requirement, which is controlled by the penalized measure $\Delta(y, y')$.

In the next chapter, we discuss the problem dealt in the thesis: Dependency Parsing. We focus on graph-based dependency parsing, which can be viewed as EBMs.

CHAPTER 3
DEPENDENCY PARSING

3.1 Introduction

In this chapter, we introduce the task of Dependency Parsing. For a sentence, dependency parsing aims to find the tree and the label for each arc of the tree, which can represent the grammatical structure of the sentence.

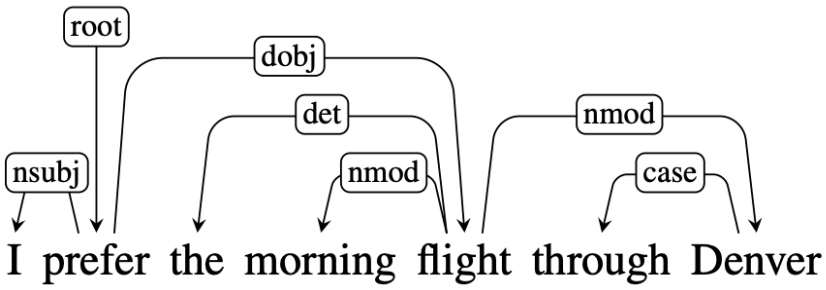

Figure 3.1: Example of Dependency Parsing

The task of Dependency Parsing can be solved with Graph-Based Models [Eisner, 1997, Kiperwasser and Goldberg, 2016, Dozat and Manning, 2017, Zhang et al., 2020a, Wang and Tu, 2020] or Transition-Based Models [Covington, 2001, Nivre, 2003, Zhang and Nivre, 2011, Choi and McCallum, 2013, Dyer et al., 2015, Zhou et al., 2015]. As is mentioned in the end of chapter 2, Graph-based dependency parsing can be viewed as EBMs. Customarily, instead of minimizing the energy for the pair (x, y), with x the sentence, y the corresponding tree, graph-based dependency parsing seeks to maximize the score for the pair (x, y), which we note as $S(x, y; \Theta)$. Graph-based models viewed the searching space $y \in \mathcal{Y}$ as a set of directed graphs and possible methods drawn from graph theory [West et al., 2001] can be used for maximization.

Unlike graph-based dependency parsing which scores the entire tree, transition-based dependency parsing builds the tree from a left-to-right manner by using transition operations [Nivre, 2003]. The operation is chosen in a greedy manner to maximize the score of each step (score of using particular operation), which is efficient, but cannot guarantee the global optimum. Thus, transition-based dependency parsing may work worse than graph-based dependency parsing for arcs which combine two words far from each other [McDonald and Nivre, 2011].

In this thesis, we focus on graph-based dependency parsing. The simplest graph-based model for dependency parsing is arc-factored model (discussed in details in section 3.4.1).

The model calculates the score of tree by making a sum over the score of arcs, which gives a linear function to the variable y.

$$S(x, y; \Theta) = \sum_{a \in y} s_a$$

where $a \in y$ represents arcs in arborescence y, with s_a the score of the arc.

For arc-factored model, efficient polynomial inference algorithms [Eisner, 1996, 1997, McDonald et al., 2005] can be used for finding the tree which maximize the score. higher-order models [McDonald and Pereira, 2006, Koo and Collins, 2010, Koo et al., 2010] can be used to increase the performance of parsing. Besides the score of arcs, higher-order models consider the sum over scores of higher-order structures like sibling, grandchild etc (discussed in detail in section 3.4.2). Meanwhile, more complex inference algorithms needed to be used for finding the tree which maximizes the score. With the rise of deep learning methods in machine, the performance of dependency parsing has greatly improved with the use of deep neural networks. In [Kiperwasser and Goldberg, 2016], a bi-directional LSTM [Hochreiter and Schmidhuber, 1997] is used to extract feature vectors from the sentence. Even with an arc-factored model, the method shows fairly good performance. [Dozat and Manning, 2017] introduces a biaffine function to calculate the score of arcs. The results prove that biaffine has better performance in practice than simple MLPs used in [Kiperwasser and Goldberg, 2016]. In recent years, second order models Zhang et al. [2020a], Wang and Tu [2020] have been constructed with deep neural networks. The results show that the deep neural network models can still benefit from higher-order structures. Besides, the use of pretrained embeddings, especially BERT [Devlin et al., 2019, Wang and Tu, 2020] can also benefit the performance of parsing.

In section 3.2, we introduce in details the task of dependency parsing, with a formal definition of projective tree and non-projective tree. In section 3.4, arc-factored models and particular higher-order models are introduced, with a detailed introduction of the inference algorithms. We focus particularly on the inference because once the inference is solved, learning of the model (calculation of the loss function) is naturally solved for these models. In section 3.5, we introduce dependency parsing with deep learning, with a detailed introduction of feature extractors, score function calculation and techniques used to take advantage of modern GPUs.

3.2 Task Description

Dependency Parsing is the task of extracting the dependency parse tree which can represent the grammatical structure of a sentence. The parse tree (also called arborescence) is an acyclic directed graph with words in the sentence as nodes. It defines the relation between the 'head' words and 'modifier' words (a directed arc from head to modifier), with a label over each arc. Following [Dozat and Manning, 2017], once the parse tree is determined, the decision of label can be solved as a classification problem. Thus, the main discussion focuses on the parse tree.

3.2.1 Projective and Non-projective Trees

In a parse tree, words in the sentences are viewed as nodes in a graph. each node has one and only one 'head' word except the root. In practice, an auxiliary 'root' is added to the beginning of the sentence to represent head of the root word. Thus, we can have a more consistent form with all nodes has one and only one head.

The parse tree can be projective or non-projective. Visually, there is no crossing arcs for projective tree while for non-projective tree, it is possible to have crossing arcs (see Figure 3.3).

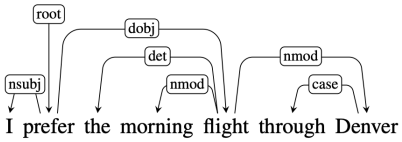

Figure 3.2: Projective Tree

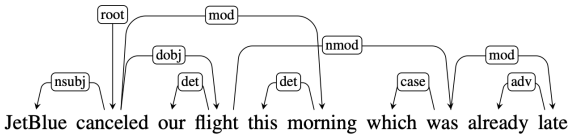

Figure 3.3: Non-projective Tree

A formal definition of projective parser tree is:

Definition 1. *An acyclic directed graph y is called projective if it satisfies the following two properties:*

1.The in-degree of every node (except the root) equals and only equals to 1.

2.$\forall(i,j) \in y$*, with* $j > i+1$*,* $\forall m$ *with* $i < m < j$*, there exists a path from* i *to* m*, which only passes nodes between* i, j *(*i, j *included).*

The second property guarantees that there exists a path from the head to any word between the head and modifier.

For non-projective tree, we only require the first property to be satisfied for the acyclic directed graph.

3.3 Notations

To facilitate the following discussions, we clarify the notations which will be used.

We note a sentence as x, with $x_1, ..., x_n$ words in the sentence, with n the number of words in the sentence. Particularly, we note x_0 the auxiliary 'root'. When the notation x is clear from the context, we only use the index to denote the word, i.e. i to represent word x_i.

For the parse tree, we use (h, m) to represent a directed arc from 'head' word x_h to 'modifier' word x_m, with $h, m \in \{0, 1, ..., n\}$. For simplicity, we use $[n]$ to represent $\{0, 1, ..., n\}$.

As parse tree is a directed tree, we can use a matrix to represent it. We use y to denote the matrix representation of a particular parse tree, with

$$y_{h,m} = \begin{cases} 1 & \text{if } (h, m) \text{ an arc in parse tree} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

For simplicity, we abuse to note $(h, m) \in y$ if (h, m) is an arc in parse tree.

As each word in the sentence has one parent, it is mandatory that $\sum_{h \in \{0,...,n\}} y_{h,m} = 1, \forall m \in \{1, ..., n\}$. This property could be seen as a basic constraint of parse trees.

We note the set of all possible parse trees for the sentence x as $\mathcal{Y}_x$, with $y \in \mathcal{Y}_x$ a particular parse tree. When x is clear in the context, we simplify it with $\mathcal{Y}$.

Similarly, we use $\mathcal{L}$ to represent the set of arc labels. The vector of arc labels in tree y is noted as $l(y) \in \mathcal{L}^n$. We note $l(y)_{hd}$ the label for arc (h, d) in y, or l_{hd} when y is clear from the context.

3.4 Graph-Based Models: From Linear to Higher-order Models

A graph-based model for dependency is basically an EBM by noting $E(x, y; \Theta) = -S(x, y; \Theta)$. The difference is that instead of minimizing the energy for correct output, customarily, we maximize the score of the sentence parse tree pair (x, y).

Different models can be constructed with different form of scoring functions, which can be divided as arc-factored model (linear model) and higher-order models. We present the details of score functions and the corresponding inference algorithms.

3.4.1 Arc-Factored Models

For arc-factored models, the score function is the sum of score of all arcs in the parse tree, i.e.

$$S(x, y) = \sum_{(h,d) \in y} s_{h,d} = \sum_{(h,d) \in [n]} s_{h,d} y_{h,d} \tag{3.2}$$

where $s_{h,d} = s(x, (h, d))$ is the score for arc (h, d). If viewing $s_{h,d}$ as a constant, the scoring function becomes a linear function to the variable y. Thus, arc-factored models are in fact linear models to the parse tree.

Inference in Arc-Factored Models

For arc-factored models, efficient inference can be realized with algorithms of polynomial complexity. For projective and non-projective cases, different algorithms can be used to ensure particular tree structures.

Inference in Projective Trees Projective trees can be efficiently decoded with Eisner algorithm ($O(n^3)$ time complexity) [Eisner, 1996],[Eisner, 1997], which is a bottom-up dynamic programming algorithm similar to CKY [Sakai, 1961, Grune and Jacobs, 2007]. It can be used to maximize score in an arc-factored model for projective trees. As Eisner algorithm is heavily used in dependency parsing, we present its basic conceptions and operations. Right and left operations are symmetric for Eisner, thus we only present the right operations.

We firstly present the conceptions of span, which is divided as complete and incomplete spans.

Definition 2. *We call $[i, j] \subseteq y$ a span, which is a sub-graph of nodes between i, j (i, j included), i.e. $[i, j] = \{(h, d) \in y | i \leq h, d \leq j\}$.*

Definition 3. *A span $[i, j]$ is called right complete if the sub-graph includes all the children of nodes between i, j (j included), with node i (the left side node) as the root of the sub-graph.*

Definition 4. *A span $[i, j]$ is called right incomplete if arc $(i, j) \in [i, j]$, and node j (the right side word) may have children nodes outside nodes between i, j (i, j included).*

Customarily, we use triangles to represent complete spans and trapezoids to represent incomplete spans (see Figure 3.4, Figure 3.5).

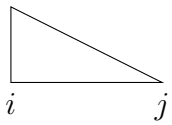

Figure 3.4: Right Complete Span

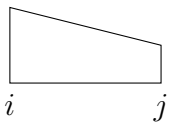

Figure 3.5: Right Incomplete Span

The (right) operations of Eisner algorithms are:

1. Construction of incomplete span with complete spans (see Figure 3.6 (a))

2. Construction of complete span with a complete span and an incomplete span (see Figure 3.6 (b))

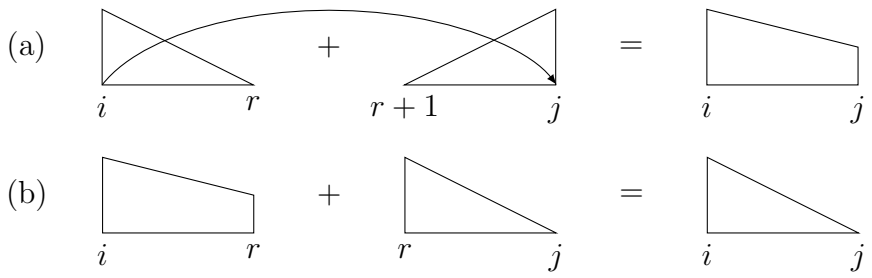

Figure 3.6: (a): Construction of Right Incomplete Span; (b): Construction of Right Complete Span

With these operations, Eisner algorithm can guarantee the projective property by constructing a right complete span including all words of the sentence (see fig 3.7) in a bottom-up way. The initial state is the complete span $[i, i], \forall i \in [n]$, i.e. we assume every node is a complete span.

Figure 3.7: Initial State and Final State

Eisner algorithm can be used to maximize the score of arc-factored model for projective tree. We use $C, I \in \mathbb{R}^{n \times n}$ to represent separately the score of complete span and incomplete span. For simplicity, we only present the construction of span from direction i to j, with $i \leq j$. The other direction can be constructed in exactly the same way.

Algorithm 1: Eisner Algorithm

1 **Initialization**: $C_{i,i} = 0, \forall 0 \leq i \leq n$;
2 **for** $m \leftarrow 1$ **to** n **do**
3 **for** $i \leftarrow 0$ **to** $n - m$ **do**
4 $j = i + m$;
5 $I_{i,j} = \max_{i \leq r < j} C_{i,r} + C_{j,r+1} + s_{i,j}$;
6 $C_{i,j} = \max_{i \leq r < j} I_{i,r} + C_{r,j}$;
7 **end**
8 **end**
9 **Return** $C_{0,n}$;

The complexity of time is $O(n^3)$ for eisner because we have two loops in line 2 and 3 of Alg 1 plus the maximization operation in line 5 and 6 for each iteration. By using the

semiring of dynamic programming algorithms [Smith, 2011, Chapter 2], we can do inference or calculate the partition function by simply changing max to argmax or logsumexp.

We mention that learning with arc-factored model can also be solved with Eisner algorithm for projective tree. For learning with max-margin loss, Eisner algorithm with argmax can be used to find the most violating tree. For learning with negative log-likelihood, Eisner algorithm with logsumexp can be used to calculate efficiently the log value of the partition function: $\log Z(x) = \log \sum_{y \in \mathcal{Y}} \exp(S(x, y))$.

Inference of Non-projective Tree For non-projective tree, inference can be realized with Chu-Liu-Edmonds algorithm (time complexity $O(n^2)$) [McDonald et al., 2005] for arc-factored model. As it is not a dynamic programming algorithm, semiring cannot be applied to calculate the partition function. In [Smith and Smith, 2007b], Matrix Tree Theorem has been shown to be applicable for calculating the partition function of non-projective tree, with also time complexity $O(n^2)$.

We mention that [Nivre and Nilsson, 2005] proposed a method to transform non-projective sentences to projective sentences. The idea is to replace non-projective arcs with projective arcs while preserving as many original arcs as possible. In order to recover the original arcs, additional information is added to the arc label, which will increase the class number of labels. Thus, a non-projective problem can now be solved as a projective problem with transformation while recovery can be done with additional information in label. Remark that firstly the recovery precision is not 100%. This is because in order to avoid the explosion of class numbers over label, the additional information is not complete. Secondly, as the number of label class increases, learning of label becomes more difficult.

3.4.2 Higher-order Models

Higher-order models can be constructed in different ways. Here we present two types of higher-order models which have efficient and accurate inference algorithms: Higher-order Structure Models for projective trees and Head Automata for non-projective trees. Remark that the method cannot be generalized to non-projective case, in which we can have at most an approximation McDonald and Pereira [2006].

Higher-order Structure Models

The first way to construct higher-order models is to consider higher-order structures. Besides linear dependency structure, more complex structures like sibling, grandparent can be considered (see Figure 3.8). Score function with higher-order structures is in fact a polynomial function with higher-order terms of variable y. By considering higher-order structures

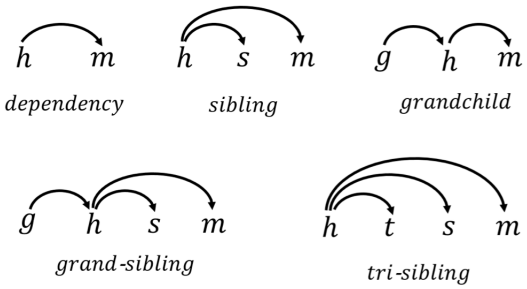

Figure 3.8: First and higher-order structures[Koo and Collins, 2010]

like sibling, grandparent, grand-sibling or tri-sibling, we introduce particular second-order and third-order terms of variable y into the score function. For example, if we consider dependency and sibling, the scoring function can be written as

$$\begin{aligned} S(x,y) &= \sum_{(h,m)\in y} s_{h,m} + \sum_{\substack{(h,m_1),(h,m_2)\in y \\ m_1\neq m_2}} s_{h,m_1,m_2} \\ &= \sum_{h,m\in[n]} s_{h,m}y_{h,m} + \sum_{h,m_1,m_2\in[n]} s_{h,m_1,m_2}y_{h,m_1}y_{h,m_2} \end{aligned} \tag{3.3}$$

while when considering grandchild, the scoring function becomes:

$$\begin{aligned} S(x,y) &= \sum_{(h,m)\in y} s_{h,m} + \sum_{(g,h),(h,m)\in y} s_{g,h,m} \\ &= \sum_{h,m\in[n]} s_{h,m}y_{h,m} + \sum_{g,h,m\in[n]} s_{g,h,m}y_{g,h}y_{h,m} \end{aligned} \tag{3.4}$$

Both equations are second-order functions to the variable y. Similar construction can be done for other higher-order relations like grandchild, grand-sibling or tri-sibling.

In [Koo and Collins, 2010], efficient dynamic programming algorithms based on Eisner have been proposed for particular structures: sibling (2^{nd} model, time complexity $O(n^3)$), grandchild (2^{nd} model, time complexity $O(n^4)$), grand-sibling (3^{rd} model, time complexity $O(n^4)$), grand-sibling and tri-sibling (3^{rd} model, time complexity $O(n^4)$). Here we present in detail the inference of sibling and grandchild. The inference for grand-sibling and tri-sibling are based on the inference of sibling and grandchild with small modifications. We note that the method is adaptable for projective tree. For non-projective tree with second order structures, we cah have approximate inference [McDonald and Pereira, 2006].

Inference with Siblings As is shown in equation (3.3), the structure sibling concerns

the tuple (h, m_1, m_2), with arc $(h, m_1), (h, m_2) \in y$, i.e. we consider two arcs which have the same head. Moreover, to ensure the existence of dynamic programming algorithms, siblings here are restricted as adjacent siblings, which requires arcs $(h, m_1), (h, m_2)$ to be consecutive arcs in the same side of word x_h. To incorporate siblings, we add the sibling span represented with rectangle: A sibling span $[i, j]$ represents adjacent modifiers of some head in position i

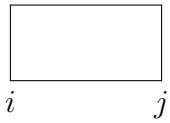

Figure 3.9: Sibling Span

and j. **explain a little more the sibling span, explain the intuition of using it** The diagram of operations with variant Eisner algorithm can be expressed as:

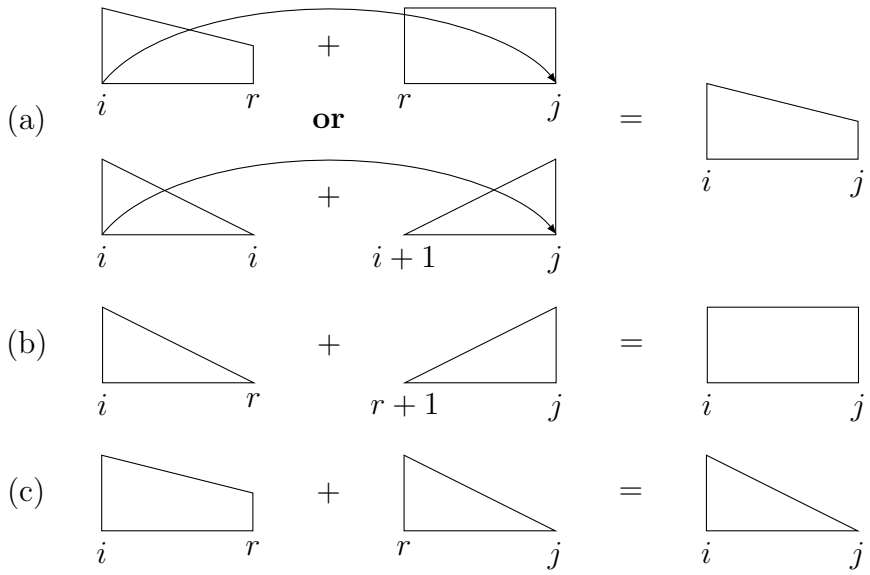

Figure 3.10: Diagrams of Variant Eisner Algorithm with Sibling. (a)Construction of right incomplete span; (b)Construction of sibling span; (c)Construction of right complete span

For the variant of Eisner algorithm which incorporate sibling span, we add $S \in \mathbb{R}^{n \times n}$ to represent the score of sibling and the algorithm can be written in similar way as Eisner (see Alg 2)

Inference with Grandchild Similar as sibing, a tuple (g, h, m) is considered for grandchild, with arcs $(g, h), (h, m) \in y$. To incorporate grandchild, instead of adding new spans like what we do in sibling, grandchild requires direct modification of complete and incomplete spans. In Figure 3.11 and Figure 3.12, we present right complete and incomplete spans

Algorithm 2: Variant Eisner Algorithm with Sibling

Initialization: $C_{i,i} = 0, \forall 0 \leq i \leq n$;
for $m \leftarrow 1$ **to** n **do**
  for $i \leftarrow 0$ **to** $n - m$ **do**
    $j = i + m$;
    $I_{i,j} = \max(\max_{i \leq r < j} I_{i,r} + S_{r,j} + s_{i,r,j} + s_{i,j}, C_{i,i} + C_{j,i+1} + s_{i,j})$;
    $S_{i,j} = \max_{i \leq r < j} C_{i,r} + C_{j,r+1}$;
    $C_{i,j} = \max_{i \leq r < j} I_{i,r} + C_{r,j}$;
  end
end
Return $C_{0,n}$;

with grandchild. Note that grandchild can be in either outer left or outer right position of the span.

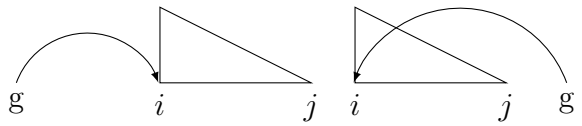

Figure 3.11: Right Complete Span with Grandchild

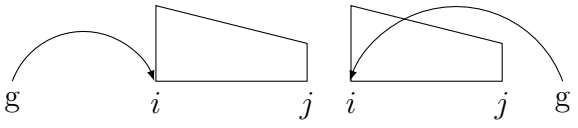

Figure 3.12: Right Incomplete Span with Grandchild

Diagrams for operations of Variant Eisner Algorithm with Grandchild is quite similar to the operations of Eisner, except that for grandchild, we need to consider an additional index g:

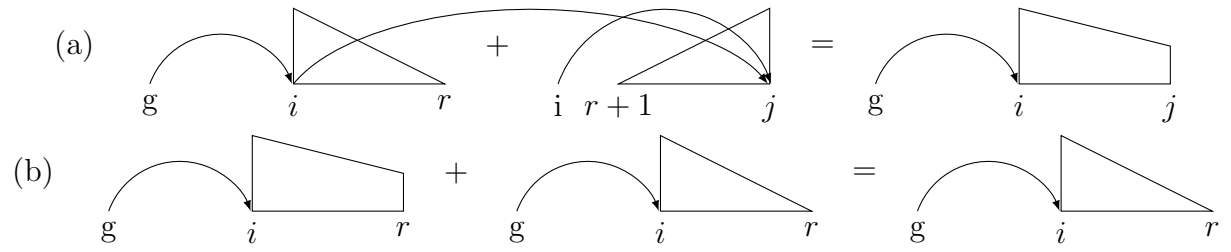

Figure 3.13: Diagrams of Variant Eisner Algorithm with Grandchild. (a)Construction of right incomplete span; (b)Construction of right complete span

Based on the new complete and incomplete spans, scores for complete and incomplete spans are represented as $C, I \in \mathbb{R}^{n \times n \times n}$, where $C^g_{i,j}, I^g_{i,j}$ represents score of complete and incomplete spans $[i, j]$ with g as the index in grandchild tuple (g, h, m). Thus, the variant Eisner algorithm with grandchild can be written in similar way as Eisner, except that we

need to add a loop for ranging with the index g. Thus, time complexity goes up to $O(n^4)$ for grandchild (see Alg 3).

Algorithm 3: Variant Eisner Algorithm with Grandchild

Initialization: $C_{i,i} = 0, \forall 0 \leq i \leq n$;
for $m \leftarrow 1$ **to** n **do**
  for $i \leftarrow 0$ **to** $n - m$ **do**
    $j = i + m$;
    for $g \leq i$ *or* $g \geq j$ **do**
      $I^g_{i,j} = \max_{i \leq r < j} C^g_{i,r} + C^i_{j,r+1} + s_{g,i,j} + s_{i,j}$;
      $C^g_{i,j} = \max_{i \leq r < j} I^g_{i,r} + C^i_{r,j}$;
    end
  end
end
Return $C_{0,n}$;

More complex higher-order structures can be viewed as a combination of sibling and grandchild and more complex variants of Eisner algorithm can be developed based on the previous modifications.

The advantage of higher-order structure model is that by considering particular higher-order structures, efficient dynamic programming algorithms can be used for efficient and exact inference. However, the advantage is in other way its limitation. Only particular higher-order structures could be considered. Even for sibling, precisely only adjacent siblings (siblings which are neighbours to each other) can be used to guarantee the existence of efficient dynamic programming algorithm. Although the first order structure dependency can be combined with higher-order structures without difficulty. combing freely higher-order structures is impossible or not easy to realize. For example, We can use either sibling or grandchild with different dynamic programming algorithms with first order dependency. However, it is not easy to use in the same time sibling and grandchild as separate structures in the same model[1]. Thus, higher-order model of this type is efficient, but quite limited in the usage of higher-order structures.

Head Automata

[Koo et al., 2010] introduces higher-order relations by using head automata. For sibling

1. grand-sibling in[Koo and Collins, 2010] connects sibling and grandchild, which forms a new structure

model, they assume that the energy function can be written as:

$$S(x, y) = \sum_{i=1}^{n} s_i(x, y_{|i}) \tag{3.5}$$

where $y_{|i} = \{y_{ij}, j = 1, ..., n, j \neq i\}$, i.e. all possible modifiers to head word x_i. The score of the parser is calculated as the sum over score of words with its all possible modifiers.

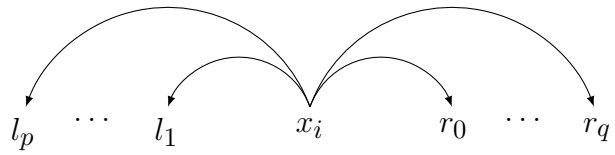

Figure 3.14: Head Automata

In order to ensure the sibling decomposition assumption, the score function is designed to have specific form:

$$s_i(x, y_{|i}) = \sum_{k=1}^{p+1} g_L(i, l_{k-1}, l_k) + \sum_{k=1}^{q+1} g_R(i, r_{k-1}, r_{k+1})$$

where $l_1, ..., l_p$, $r_1, ..., r_q$ are left and right modifiers to the word x_i under $y_{|i}$. $l_0 = r_0 =$ START is the initial state and $l_{p+1} = r_{q+1} =$ END is the end state. s_i represents the total score of left/right modifiers of x_i with adjacent sibling scores.

With this form of score function, a variant of Viterbi algorithm [Jurafsky, 2000] can be used to calculate the the best set of modifiers with sibling scores for each x_i in time complexity $O(n^2)$. The problem is that the solution only satisfies the single head property. We can not guarantee it to be acyclic or projective. To solve this problem, dual decomposition [Lemaréchal, 2001] can be used to ensure particular tree structure by considering the Integer Linear Programming (ILP) problem:

$$\begin{aligned} &\underset{z \in \mathcal{Z}, y \in \mathcal{Y}}{\operatorname{argmax}} \; f(x, z, y) = S(x, z) + h(x, y) \\ &\text{s.t. } z = y \end{aligned}$$

where $\mathcal{Y} \subseteq \mathcal{Z}$, with $\mathcal{Z}$ the set for solutions satisfying the single head property. $h(y) = \sum_{(h,d) \in y} \gamma_{h,d}(x) y_{h,d}$ is the score of an arc-factored model, with which we can well guarantee tree structure.

3.5 Graph-Based Dependency Parsing with Deep Neural Networks

The use of deep neural networks has greatly improved the performance of dependency parsing. In this section, we present the construction of neural networks for the feature extraction (section 3.5.1) and the calculation of the score function (section 3.5.2). Algorithms for inference, although directly usable, can be modified to benefit from modern GPUs. We present in the end an end-to-end learning method, which gives approximate inference, but can still benefit from higher-order structures with neural networks.

3.5.1 Feature Extraction with Bi-directional LSTM

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN). LSTM is mostly used in NLP and has shown strong capacity to capture not only a single data point (a word), but the whole sequence of data (a sentence).

For a sequence of data $x_1, ..., x_n$, LSTM calculates the hidden state for each step with the following equations:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

where $x_t \in \mathbb{R}^d$ is the input vector; $f_t, i_t, o_t \in (0,1)^h$ are called separately the forget, input, output gate's activation vector; $\tilde{c}_t \in (-1,-1)^h$ is the cell input activation vector; $c_t \in \mathbb{R}^h$ is the cell state vector; $W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h$ are learnable parameters; and h_t is the hidden state (output vector of LSTM unit). σ_g is the sigmoid function and σ_c, σ_h are hyperbolic tangent functions.

The diagram of LSTM is shwon in Figure 3.15. For each step, a LSTM unit uses hidden vector and cell state vector in the previous step, combined with the input vector to generate hidden vector and cell state for the next step. Thus it is naturally adapted for treating data in form of sequence.

When using LSTM to generate the hidden vector $\overrightarrow{h_t}$ to the order of a sentence, $\overrightarrow{h_t}$ is dependent to words $x_1, ..., x_t$ while it is not dependent to words $x_{t+1}, ..., x_n$. However, if we

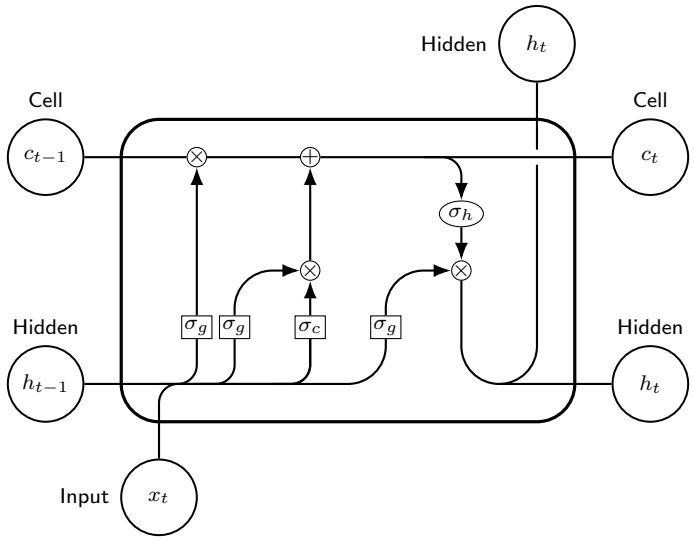

Figure 3.15: LSTM Unit

use LSTM to generate hidden vector $\overleftarrow{h_t}$ in inverse order, $\overleftarrow{h_t}$ is dependent to words $x_n, ..., x_t$ while it is not dependent to words $x_{t-1}, ..., x_1$. To make the output of each state dependent on the whole sentence, a concatenation of the forward hidden vector and the backward vector can be used to generate the ouptut $v_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$. This gives us a bidirectional LSTM explained in Figure 3.16.

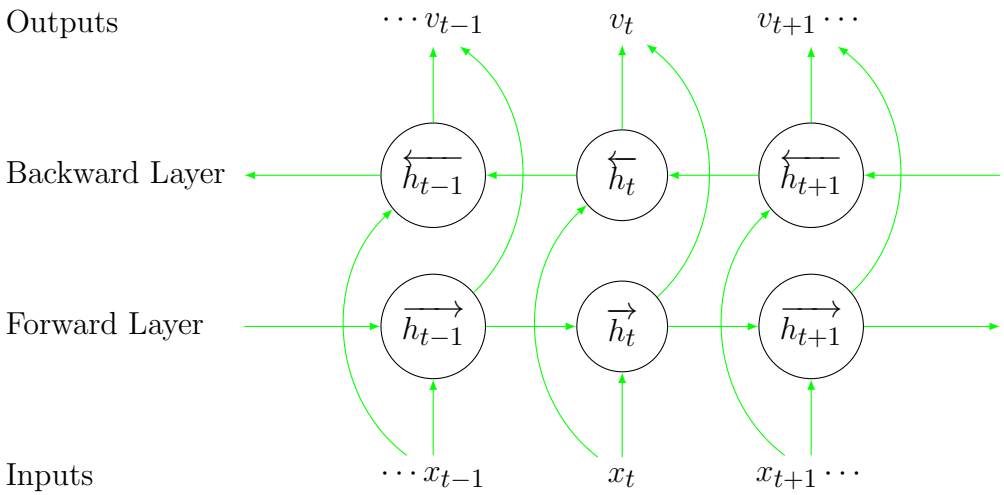

Figure 3.16: Bidirectional LSTM

Bidirectional LSTM layers can be stacked to extract better feature vectors. For example, previous works use 3 layers of Bidirectional LSTMs to extract feature vectors. [Dozat and Manning, 2017, Zhang et al., 2020a, Wang and Tu, 2020]

We mention that concatenation of pretrained embeddings like BERT [Devlin et al., 2019] and fastText [Mikolov et al., 2018] to word-embeddings can improve a lot the performance.

3.5.2 Calculation of Scores with Affine Functions

To calculate the score for an arc (h, d), a sibling part (h, d, s) or a grandchild part (g, h, d), we can use firstly MLPs to generate different features vectors for head, modifier, sibling or grandchild. For example, when considering arc (h, d), head and modifier vectors can be generated separately with different MLPs, i.e.

$$v_t^h = \text{MLP}^h(v_t)$$
$$v_t^d = \text{MLP}^d(v_t)$$

Similar operations can be applied to generate v_t^s and v_t^g.

The score of arc (i, j) is calculated with a biaffine function by using:

$$s_{ij} = (v_i^h)^T W^{\text{biaffine}} \begin{bmatrix} v_j^d \\ 1 \end{bmatrix} \tag{3.6}$$

where $W^{\text{biaffine}} \in \mathbb{R}^{d \times (d+1)}$ the trainable parameters of biaffine, with d the dimension of feature vector. The additional 1 is used to add a bias term for v_i^h.

For score of higher-order structures like sibling, we consider a tuple (i, j, k) and traiffine can be used:

$$s_{ijk} = \begin{bmatrix} v_k^s \\ 1 \end{bmatrix}^T (v_i^h)^T W^{\text{traiffine}} \begin{bmatrix} v_j^d \\ 1 \end{bmatrix} \tag{3.7}$$

where $W^{\text{triaffine}} \in \mathbb{R}^{d \times (d+1) \times (d+1)}$ the trainable parameters of triaffine.

Remark that the calculation can be efficiently done with modern GPUs, and can be well batchified.

Similar method can be applied for other higher-order structures. One problem is that the number of parameters for affine functions increase exponentially to the degree of order ($O(d^{k+1})$, with k the order degree). In practice, Calculating scores of higher-order structures with affine functions cannot be applied easily for structures with order higher than 3 due to the memory overflow problem on current GPUs with moderate memory size (16GB-32GB).

3.5.3 Batchified Dynamic Programming Algorithms

As is show in Alg 2, Variant Eisner Algorithm with sibling has time complexity $O(n^3)$. Zhang et al. [2020a] shows that one loop can be well batchified. Thus the capacity of modern GPUs in parallel computing, we can increase a lot the speed. The batchified algorithm with sibling is shown in Alg 4, in which the inner loop in line 3 (see Alg 1) is batchified.

Algorithm 4: Batchified Eisner Algorithm with Sibling

1 **Initialization**: $C_{i,i} = 0, \forall 0 \leq i \leq n$;
2 **for** $m \leftarrow 1$ **to** n **do**
3 | **Batchify** $0 \leq i; j + m \leq n$;
4 | $I_{i,j} = \max(\max_{i \leq r < j} I_{i,r} + S_{r,j} + s_{irj} + s_{ij}, C_{i,i} + C_{j,i+1} + s_{ij})$;
5 | $S_{i,j} = \max_{i \leq r < j} C_{i,r} + C_{j,r+1}$;
6 | $C_{i,j} = \max_{i \leq r < j} I_{i,r} + C_{r,j}$;
7 **end**
8 **Return** $C_{0,n}$;

The same technique can be used for decoding and calculating the log partition function, with max replaced with argmax or logsumexp.

3.5.4 Gradient Based MBR Decoding

[Smith and Smith, 2007b] propose to use Minimum Bayes-risk (MBR) decoding for dependency parsing. The idea is that instead of using algorithms like Eisner to maximize directly the score function, we can firstly calculate the marginal probability of arc:

$$p((h,d)|x) = \sum_{\substack{y \in \mathcal{Y} \\ (h,d) \in y}} p(y|x)$$

Then we change inference to maximize the probability of choosing correct arcs, which gives a less risky solution without low probability arcs:

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} \prod_{(h,d) \in y} p((h,d)|x)$$

The equation is equivalent as:

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{(h,d) \in y} \log p((h,d)|x)$$

Thus, we can view $\log p((h,d)|x)$ as the score of arc and treat it as an arc-factored model, with Eisner or Chu-Liu-Edmonds for inference.

One problem of MBR is how to calculate efficiently the marginal probability of arc. Even for models which can be solved with Eisner or its variants (inside algorithm), an outside algorithm needed to be used for the calculation of the marginal probability. [Eisner, 2016] proves that the outside can be replaced with back-propagation. Inspired by the proof, [Zhang et al., 2020a] shows that:

$$p((h,d)|x) = \frac{\partial \log Z(x)}{\partial y_{h,d}}$$

Thus, the marginal probability can be efficiently calculate with back-propagation. MBR decoding has shown to achieve a tiny but consistent augmentation in performance in Zhang et al. [2020a].

For arc-factored model (either arc-factored, or with MBR), inference with Eisner can also be replaced with back-propagation. We use the MBR as an example. For naturally arc-factored model, we just need to replace $\log p((h,d)|x)$ with the true score value of arc:

Firstly, run Eisner over $\log p((h,d)|x)$ to get the maximum log probability, noted as $\log p(y|x)$.

Then calculate

$$y_{h,d} = \frac{\partial \log p(y|x)}{\partial \log p((h,d)|x)}$$

Then $y_{h,d}$ is directly the matrix representation of the projective tree. This is because the gradient of max is one for selected score and 0 for non-selected score in back-propagation[2]. Thus, calculating the partial gradient of the max score with the score of arc gives directly the matrix representation of the tree.

3.5.5 End-to-End Learning with MFVI

The previous methods consider the case where exact dynamic programming algorithms exist. However, this limits the model from using higher-order structures. For example, there exists variants of Eisner algorithm for either siblings or grandchildren combining with first order dependency, but not when using the two together.

[Wang and Tu, 2020] consider exactly the model with dependency, sibling and grandchild. Although efficient algorithms do not exist, approximate method like Mean Field Variational Inference (MFVI) [Fox and Roberts, 2012] can be used to estimate the marginal probability of arc. The idea is to use a simple $p_{h,d}$ to approximate the true marginal distribution of the

2. It is not mathematically strict, but back-propagation in pytorch is set in this way

arc. The simple distribution satisfies $\sum_{h=0}^{n} p_{h,d} = 1, \forall h$ to meet the single head property. The mean field hypothesis requires that the distributions for every column d are independent.

We note the marginal distribution of arc at step t as $p^{(t)}((h,d))$ or simply $p^{t}_{h,d}$ when x is clear in the context. The update rule of MFVI with sibling and grandchild is:

$$p^{(t)}_{h,d} \propto s_{h,d} + \sum_{k} p^{(t-1)}_{h,k} s^{\text{sib}}_{h,d,k} + p^{(t-1)}_{d,k} s^{\text{gp}}_{h,d,k} + p^{(t-1)}_{k,h} s^{\text{gp}}_{k,h,d}$$

Intuitively, the right part can be viewed as a sum of score of all structures which have arc (h,d).

As for every column d, we require there exits only one arc. Thus, we can normalize the probability by using softmax:

$$p^{(t)}_{h,d} = \frac{\exp(s_{h,d} + \sum_k p^{(t-1)}_{h,k} s^{\text{sib}}_{h,d,k} + p^{(t-1)}_{d,k} s^{\text{gp}}_{h,d,k} + p^{(t-1)}_{k,h} s^{\text{gp}}_{k,h,d})}{\sum_{h'} \exp(s_{h',d} + \sum_k p^{(t-1)}_{h',k} s^{\text{sib}}_{h',d,k} + p^{(t-1)}_{d,k} s^{\text{gp}}_{h',d,k} + p^{(t-1)}_{k,h'} s^{\text{gp}}_{k,h',d})} \tag{3.8}$$

With the framework of SPEN, p^0 can be initialized with the local (linear) part of the model and Wang and Tu [2020] shows that 3 iterations of MFVI is sufficient to benefit the model from higher-order structures.

We note that the implementation of MFVI in Wang and Tu [2020] does not guarantee the convergence to a local optimum because they apply MFVI for all columns in the same time, while exact MFVI should be applied in a column by column way. Still training is stable for that a small number of iterations 3 is used, which does not makes big change over the original distribution.

3.6 Conclusion

In this chapter, we present the basis of dependency parsing with graph-based models.

Dependency parsing aims to find the parsing tree, which can represent the grammatical structure of the sentence. The parse tree can be classified as projective tree and non-projective tree. Visually, no crossing arcs exist in projective tree while non-projective tree can have arcs. We give also a formal definition of these two types of parse tree.

Arc-factored model and specific higher-order models, which have exact and efficient algorithms, are presented. We present in details the inference algorithms for these models. For projective tree, once the inference problem is solved, the learning problem (calculate the loss function) is naturally solved under the framework of EBMs. For non-projective tree, inference can be solved with chu-liu-edmond [McDonald et al., 2005]. Learning with hinge

loss is naturally solved. However, learning with the negative log-likelihood cannot be solved with the same algorithm. Instead, matrix tree theorem [Smith and Smith, 2007b] can be used for estimating the partition function with the same time complexity.

Deep learning, especially the use of neural network has greatly improve the performance of dependency parsing. We introduce the feature extractor, calculation of the score function with neural networks in parsing. The inference algorithms are not changed, but can be modified to benefit from the modern GPUs. We present the batchified version of Eisner algorithm and MBR decoding with back-propagation. An end-to-end model with approximate inference is presented in the end. Although with a non-exact estimation of the probability of arc, the deep learning model still works and can benefit from higher-order structures.

CHAPTER 4
MIXTURE-OF-EXPERTS FOR DEPENDENCY PARSING

4.1 Introduction

In this chapter, we present the mixture-of-experts (MoE) for dependency parsing. MoE has the potential to approximate any non-linear model with sufficient simple experts. Thus, we can use basic arc-factored and second-order models presented in Chapter 3 as expert to construct complex MoE models.

Combinations of elementary parsers are known to improve accuracy. Sometimes called *joint systems*, they often use different representations, *i.e.* lexicalized constituents and dependencies [Rush et al., 2010, Green and Žabokrtský, 2012, Le Roux et al., 2019, Zhou et al., 2020]. These approaches have been devised to join the strengths and overcome the weaknesses of elementary systems.

In this chapter, however, we follow another line of research consisting of mixtures and products of similar experts [Jacobs et al., 1991, Brown and Hinton, 2001], instantiated for parsing in [Petrov et al., 2006, Petrov, 2010] and especially appealing when individual experts have high variance, typically when training involves neural networks. Indeed [Petrov, 2010] used products of experts trained via Expectation-Maximization (a non-convex function minimization) converging to local minima.

We propose to study the combination of parsers, from a probabilistic point of view, as a mixture model, *i.e.* a learnable convex interpolation of probabilities. This has previously been studied in [Petrov et al., 2006] for PCFGs with the goal of overcoming the locality assumptions, and we want to see if neural graph-based dependency parsers, with non-markovian feature extractors, can also benefit from this framework. It has several advantages: it is conceptually simple and easy to implement, it is not restricted to projective dependency parsing (although we only experiment this case), and while the time and space complexity increases with the number of systems, this is hardly a problem in practice thanks to GPU parallelization.

Simple averaging models, or ensembles, can also be framed as mixture models where mixture coefficients are equal. We are able to quantify the variance reduction, both theoretically and empirically and show that this simple model of graph-based parser combinations perform better on average, and achieve a higher accuracy than single systems.

While the full mixture model is appealing, since it could in principle both decrease variance and find the optimal interpolation weights to better combine parser predictions, the non-convexity of the learning objective is a major issue that, when added to the non-convexity of potential functions, can prevent parameterization to converge to a good solution.

By trying to specialize parsers to specific input, the variance is not decreased. More importantly, experiments indicate that useful data, that is data with an effect on parameterization, becomes too scarce to train the clustering device.

Another drawback of finite mixture models is that inference, *i.e.* finding the optimal tree, becomes intractable. We tackle this issue by using an alternative objective similar to Minimal Bayes-Risk [Goel and Byrne, 2000] and PCFG-LA combination [Petrov, 2010] for which decoding is exact.

The contributions can be summarized as follows:

- We frame dependency parser combinations as finite mixture models (Section 4.2) and discuss two properties: averaging and clustering. We derive an efficient decoder (LMBR) merging predictions at the arc level (Section 4.3).
- When isolating the averaging effect, we show that resulting systems exhibit an empirical variance reduction which corroborates theoretical predictions, and are more accurate (Section 4.4).
- We study the causes of instability in mixture learning, outline why simple regularization is unhelpful and give an EM-inspired learning method preventing detrimental over-specialization (Section 4.5). Still, improvement over mere averaging is difficult to achieve.
- These methods obtain state-of-the-art results on two standard datasets, the PTB and the CoNLL09 Chinese dataset (Section 5.7), with low variance making it robust to initial conditions.

4.2 Mixture of Experts

4.2.1 Parsers as Experts

Experts can be any probabilistic graph-based dependency parser, provided that we can efficiently compute the energy of a parse tree, the global energy of a sentence (the sum of all parse tree energies, called the partition function) and the marginal probability of an arc in a sentence.

In the remaining we focus on projective first- and second-order parsers, where these quantities are computed via tabular methods or backpropagation[1].

1. Matrix-tree theorem could be used to adapt this work to non-projective first-order models [Smith and Smith, 2007a]

Tree structure For a graph-based dependency parser, the tree probability is defined as:

$$p(y|x) = \frac{\exp(s(x,y))}{Z(x) \equiv \sum_{y' \in \mathcal{Y}(x)} \exp(s(x,y'))}$$

with $s(x,y)$ the tree energy giving the correctness of y for x, and $Z(x)$ the partition function.

In first-order models [Eisner, 1996], tree scores are sums of arc scores:

$$s(x,y) = \sum_{(h,d) \in y} s(h,d)$$

Eisner [1997] generalizes scores to the second-order by considering pairs of adjacent siblings:

$$s(x,y) = \sum_{(h,d) \in y} s(h,d) + \sum_{\substack{(h,d_1) \\ (h,d_2) \in y}} s(h,d_1,d_2)$$

with $h < d_1 < d_2$ or $d_2 < d_1 < h$. For projective first- or second-order models, $Z(x)$ and $p(y|x)$ are efficiently calculated [Zhang et al., 2020b]. Moreover marginal arc probability $p\big((h,d)|x\big)$ can be efficiently calculated from the partition function by applying backpropagation from $\log Z(x)$ to $s(h,d)$, see[Eisner, 2016, Zmigrod et al., 2020, Zhang et al., 2020a]:

$$p((h,d)|x) = \sum_{\substack{y \in \mathcal{Y}(x) \\ (h,d) \in y}} p(y|x) = \frac{\partial \log Z(x)}{\partial s(h,d)}$$

Tree Labelling The labelling model is also a Boltzmann distribution:

$$p(l|(h,d),x) = \frac{\exp(s(l,h,d))}{\sum_{l' \in \mathcal{L}} \exp(s(l',h,d))}$$

where $s(l,h,d)$ is the score for label l on (h,d).

Following[Dozat and Manning, 2017, Zhang et al., 2020a], label predictions are independent:

$$p(l(y)|y,x) = \prod_{(h,d) \in y} p(l_{hd}|(h,d),x) \tag{4.1}$$

Parse Probability Given the structure y and its labelling $l(y)$, the parse probability is:

$$p\big(l(y),y|x\big) = p(y|x) \times p\big(l(y)|y,x\big) \tag{4.2}$$

Learning Potential functions s can be implemented by feed-forward neural networks or biaffine functions [Dozat and Manning, 2017], and parameterized by maximizing a log-likelihood.

4.2.2 Mixture and Averaging

For arborescence probabilities a finite mixture model (MoE) is a weighted sum of the probabilities given by all experts:

$$p(y|x) = \sum_{k=1}^{K} \omega_k(x) p_k(y|x) \tag{4.3}$$

where mixture weights verify $\forall x, \omega_k(x) \geq 0$ and $\sum_{k=1}^{K} \omega_k(x) = 1$ and can be adjusted by a gating network [Jacobs et al., 1991]. We can interpret ω as a device whose role is to *cluster* input in K categories and assign each category to an expert.

By forcing $\omega_k(x) = \frac{1}{K}, \forall x$, we have a simpler averaging model, sometimes called *ensemble*:

$$p(y|x) = \frac{1}{K} \sum_{k=1}^{K} p_k(y|x)$$

Note that MoEs combine elementary probabilities, not tree scores: each expert energy is first normalized before the combination.

A similar mixture is applied to labelling, *i.e.*:

$$p(l(y)|y, x) = \sum_{k=1}^{K} \lambda_k(x) p_k(l(y)|y, x)$$

4.3 Decoding with a Mixture Model

Learning MoEs will be covered in Section 4.5 and we first turn to the problem of finding an appropriate tree, for instance the most probable parse tree:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} p(y|x) = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \sum_{k=1}^{K} \omega_k(x) p_k(y|x)$$

This maximization is difficult, even in the absence of labels, since this isn't a log-linear function of the arc scores anymore: y^* cannot be searched in the log-space among unnormalized arc scores.

4.3.1 MBR Decoding

In this case, a more attractive alternative is Minimum Bayesian Risk (MBR) decoding [Smith and Smith, 2007a], because it decomposes error in a way similar to the metrics used in dependency parsing (UAS/LAS) and is tractable. MBR requires to compute marginal arc probabilities which are the weighted sums of elementary marginals:

$$p\big((h,d)|x\big) = \sum_{k=1}^{K} \omega_k(x) p_k\big((h,d)|x\big)$$

The intuition behind MBR is that instead of maximizing the probability of the parse tree, we try to minimize the risk of choosing wrong arcs, *i.e.* to maximize the arc marginals in the parse tree:

$$y^* = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}} \prod_{(h,d) \in y} p\big((h,d)|x\big) = \text{MBR}(x)$$

Once computed marginal log-probabilities, Eisner algorithm [Eisner, 1996], [Eisner, 1997] or Chu-Liu-Edmonds [McDonald et al., 2005] can be applied to solve MBR.

4.3.2 MBR Decoding with Labels

In many dependency parsing models, decoding of arcs and labels is pipelined, see for instance[Dozat and Manning, 2017, Zhang et al., 2020a, Fossum and Knight, 2009]: first arcs are decoded and then, with the decoded arcs, maximization is performed over labels:

$$y^* = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}}\, p(y|x) \text{ then } l^* = \underset{l = l(y^*)}{\operatorname{argmax}}\, p(l|y^*, x)$$

However, solutions found this way are not the maximizers for $p(l, y|x)$, as defined in Eq. 4.2. The problem is that the effect of labelling is not considered in arc decoding: a high probability arc can get picked up even with a low label score.

First we remark that each label in l^* is the most probable label l for a pair (h, d), denoted by L_{hd}. Decoding becomes:

$$y^* = \underset{y}{\operatorname{argmax}}\, p(y|x) \prod_{(h,d) \in y} p(L_{hd}|(h,d), x)$$

This way l^* is deterministic wrt to y^* and (y^*, l^*) are maximizers for Eq. 4.2. We note labelling $L(y)$ where $l(y)_{hd} = L_{hd}, \forall (h,d) \in y$. This can be combined with MBR without

changing decoding algorithms, and we call this variant LMBR:

$$y^* = \underset{(y,l=L(y))}{\operatorname{argmax}} \prod_{(h,d)\in y} p\big((h,d)|x\big)p\big(L_{hd}|(h,d),x\big)$$

i.e. we can apply MBR with arc probabilities reparameterized with label probabilities. Experiments show that LMBR exhibits a small but consistent accuracy increase over MBR.

4.4 Averaging and Variance Reduction

In this section we assume all experts to be equally weighted. We define the variance of the system on $\mathcal{T}$ as the average variance of marginal arc probability:

$$\sigma^2 = \frac{\sum_{(x,y)\in\mathcal{T}} \sum_{(h,d)\in y} \sigma^2\Big(p\big((h,d)|x\big)\Big)}{\sum_{(x,y)\in\mathcal{T}} |y|}$$

with $\sigma^2(p((h,d)|x))$ the variance of the marginal probability.

We show how the variance of the MoE is smaller than the variance of experts. We focus on structure prediction $p(y|x)$, but definitions are applicable to the labelling model as well. This is an already known result for mixture models in general, but the proof is here instantiated for a mixture of graph-based parsers. Moreover, we will recover this result experimentally in Section 5.7.

Assuming we have a mixture of K elementary systems, we will estimate the marginal probability variance with:

$$\sigma^2\Big(p\big((h,d)|x\big)\Big) = \frac{1}{K}\sum_{k=1}^{K}(\pi(k)-\bar{\pi})^2$$

with $\pi(k)$ the probability $p_k\big((h,d)|x\big)$ given by the k^{th} elementary system and average $\bar{\pi} = \frac{1}{K}\sum_{k=1}^{K}\pi(k)$

Increasing the number of experts in the MoE will decrease variance of the system. To see this, we assume that the marginal probability for a well trained expert, over a fixed sentence and a fixed arc, is a measurable function $f_{(h,d),x} : \mathbb{R} \to \mathbb{R}$ of a random seed $S_k \in \mathbb{R}$, which represents the fact that p_k is the result of a learning process with many sources of

randomization[2] (initialization, stochastic batches, dropout...):

$$p_k((h,d)|x) = f_{(h,d),x}(S_k)$$

with $S_k \in \mathbb{R}$ a random seed assigned to k^{th} expert at the beginning of training, assumed to be independent for different experts.

Since in practice a pseudo-random generator is used, the value of marginal probability for particular sentence and arc is deterministic when the random seed is fixed. Thus, it is sufficient to use a deterministic function to represent $p_k((h,d),x)$, with random seed S_k as input. Moreover, we just need the function to be measurable.

We can now view $f_{(h,d),x}(S)$ as a random variable and we note its variance as $\sigma^2_{(h,d),x}$. It is in fact the variance of the marginal arc probability given by this expert, for (h,d) given x. For an averaging MoE, the marginal probability becomes:

$$p((h,d)|x) = \frac{1}{K}\sum_{k=1}^{K} f_{(h,d),x}(S_k)$$

with K number of experts in the mixture model.

If random variables $\{S_k\}_{k\in K}$ are independent, $\{f_{(h,d),x}((S_k)\}_{k\in K}$ also are independent [Baldi, 2017]. Thus, the variance of the mixture model for particular sentence and arc should be $\frac{1}{K}$ times the variance of experts:

$$\Sigma^2_{(h,d),x} = \frac{\sigma^2_{(h,d),x}}{K} \tag{4.4}$$

with Σ the variance of the mixture model. In other words, the log-variance of a mixture model decreases linearly with $\log K$, with slope -1, *i.e.*:

$$\log \Sigma^2_{(h,d),x} = \log \sigma^2_{(h,d),x} - \log K$$

Experiments in Section 5.7 Figure 4.1 show that the estimated log-variance of the averaging system decreases when the number of experts increases and that this relation is close to linear with a slope approaching -1, comforting our independence assumption.

2. f should also be indexed by the training set, but we omit this for the sake of readability.

4.5 Training with Clustering

When mixture weights are adjustable, MoE models are able to give more credit to experts believed to perform better on specific input. This can be exploited during parameterization. The role of ω is thus to learn how to cluster input into K categories, each category being assigned to an expert.[3]

For input sentence x and corresponding tree y, assuming parameterization is performed by maximizing the log-likelihood of the training set via SGD, the objective of mixture model learning with gating network ω can be written as:

$$L(\phi, \theta) = \log \sum_{k=1}^{K} \omega_k(x; \phi) p_k(y|x; \theta_k) \tag{4.5}$$

where ϕ are the parameters of the gating network, and θ_k are the parameters of the k^{th} expert.

Partial derivatives to the gating network are:

$$\frac{\partial L(\phi, \theta)}{\partial \phi} = \sum_{k=1}^{K} \frac{\omega_k(\phi) p_k(\theta_k)}{\sum_{k'=1}^{K} \omega_{k'}(\phi) p_{k'}(\theta_{k'})} \frac{\partial \log \omega_k(\phi)}{\partial \phi} \tag{4.6}$$

while for expert parameters we have:

$$\frac{\partial L(\phi, \theta)}{\partial \theta_k} = \frac{\omega_k(\phi) p_k(\theta_k)}{\sum_{k'=1}^{K} \omega_{k'}(\phi) p_{k'}(\theta_{k'})} \frac{\partial \log p_k(\theta_k)}{\partial \theta_k}. \tag{4.7}$$

We found that optimizing directly with equations (4.6) and (4.7) causes degeneration, *i.e.* one ω_k approaches 1 while the other $\omega_{k'}$ decrease to almost 0. Indeed, gradient ascent with (4.6) will increase ω_k for an expert k that gives high weight to training samples while gradient ascent with (4.7) will generate increased gradient, and in turn increased probabilities, for experts with high value of ω_k. The two processes re-enforce each other and result quickly in an extreme partition between experts.

One may think that the degeneration problem can be alleviated with a smoothing prior or regularization. In practice, we tried entropy as regularization to force towards a uniform distribution on ω_k. We found that a heavy entropy penalization is required to avoid the degeneration problem, which makes ω_k too uniform to be an accurate clustering device.

3. We note that averaging MoE models do not require a specific training: experts can be trained separately and the ensemble is gathered at decoding time only.

Avoid Extreme Partition Thus, to alleviate the degeneration problem without forcing a strong smoothing constraint, we propose to modify Eq. (4.6) into:

$$\frac{\partial L^{'}(\phi,\theta)}{\partial \phi} = \sum_{k=1}^{K} \frac{p_k(\theta_k)}{\sum_{k'=1}^{K} p_{k'}(\theta_{k'})} \frac{\partial \log \omega_k(\phi)}{\partial \phi} \tag{4.8}$$

i.e. we force the weight update to be proportional to the relative probability. The advantage of Eq. (4.8) is that gradient are weighted by a more objective quantity $\frac{p_k(\theta_k)}{\sum_{k'=1}^{K} p_{k'}(\theta_{k'})}$. For an example where $p_k(x)$ is close to uniform, we can benefit from the averaging effect, while for an example which shows strong preference for a particular expert, we can also learn the partition coefficients proportional to their correctness.

Stabilize Training Neuron dropout [Srivastava et al., 2014b] is a common technique to avoid overfitting which unfortunately proved difficult in this setting. The problem is that $s_k(x,y)$ gives very different results with or without dropout which reflects on $p_k(y|x)$ causing drastic changes from one evaluation to the other. To mitigate this problem, we use probabilities without dropout (noted as $\tilde{p}_k(\theta)$) to calculate the weighted coefficients of gradient.

The final optimization process can be separated into two alternate parts, *(i)* optimization of the gating parameters:

$$\frac{\partial L^{'}(\phi,\theta)}{\partial \phi} = \sum_{k=1}^{K} \frac{\tilde{p}_k(\theta_k)}{\sum_{k'=1}^{K} \tilde{p}_{k'}(\theta_{k'})} \frac{\partial \log \omega_k(\phi)}{\partial \phi}$$

and *(ii)* optimization of experts:

$$\frac{\partial L(\phi,\theta)}{\partial \theta_k} = \sum_{k=1}^{K} \frac{\omega(\phi)\tilde{p}_k(\theta_k)}{\sum_{k'=1}^{K} \omega_{k'}\tilde{p}_{k'}(\theta_{k'})} \frac{\partial \log p_k(\theta_k)}{\partial \theta_k}$$

In practice, this permitted reaching a lower loss value after training.

4.6 Experiments

Data We run experiments over two datasets for projective dependency parsing: The English Penn Treebank (PTB) data with Stanford Dependencies [Marcus et al., 1993] and CoNLL09 Chinese data [Hajič et al., 2009]. We use standard train/dev/test splits and evaluate with UAS/LAS metrics. Customarily, punctuation is ignored on PTB evaluation.

Experts We run tests with first-order (FOP) and second-order parsers (SOP) as mixture

model experts, with re-implemented versions of the CRF and CRF2o parsers of [Zhang et al., 2020a].[4] For decoding, we use the LMBR decoding presented in Section 4.3.2, which guarantees a small but consistent improvement over pipeline MBR decoding.

For each input word, these systems use 3 embeddings: the first is a fixed pretrained vector[5], the second is trainable and looked-up in a table, and the third is computed by a BiLSTM at the character level (CharLSTM). The first two embeddings are summed and concatenated with the char sequence embedding. For FOP and SOP, contextual lexical features are the results of 3-layer BiLSTMs applied to word embedding sequences. The scoring of arcs is then similar to[Dozat and Manning, 2017]: lexical features are transformed for head or modifier roles by two feed-forward networks and combined to score arcs via a biaffine transformation.

On PTB, in order to compare with recent parsing results, we set up BFOP and BSOP (B for Bert), variants of the FOP and SOP settings: we follow [Fonseca and Martins, 2020] and concatenate an additional BERT embedding[Devlin et al., 2019] (the average of the 4 last layers of the *bert-base-uncased* model) to the embedding vector fed to the BiLSTM layers.

Gating (mixture weights ω) is implemented by a K-class softmax over a feed-forward network whose input are the concatenation of initial and final contextual lexical feature vectors returned by the 3-layer BiLSTM. Hyper-parameters are set similarly to Zhang et al. [2020a], with the exception of the learning rate decreased to 10^{-4} and patience (that is the maximum number of epochs without LAS increase on the development set) set to 20

We train 12 independent models for each expert type, with random seed set to system time.

K	FOP		SOP	
	UAS	LAS	UAS	LAS
1	95.83 $^{\mathbf{96.04}}_{95.72}$ ±0.08	94.06 $^{94.24}_{93.91}$ ±0.08	95.87 $^{95.94}_{95.77}$ ±0.06	94.07 $^{94.16}_{93.97}$ ±0.05
2	95.88 $^{\mathbf{96.04}}_{95.76}$ ±0.06	94.15 $^{94.32}_{94.05}$ ±0.07	95.92 $^{96.05}_{95.85}$ ±0.05	94.15 $^{94.27}_{94.08}$ ±0.04
3	95.93 $^{96.03}_{95.84}$ ±0.05	94.22 $^{94.32}_{94.11}$ ±0.05	95.94 $^{96.04}_{95.85}$ ±0.06	94.18 $^{94.27}_{94.08}$ ±0.06
4	**95.95** $^{\mathbf{96.04}}_{95.90}$ ±0.04	**94.24** $^{\mathbf{94.35}}_{94.16}$ ±0.05	**95.98** $^{\mathbf{96.07}}_{95.91}$ ±0.05	94.22 $^{\mathbf{94.31}}_{94.15}$ ±0.04
5	95.93 $^{96.00}_{95.84}$ ±0.04	**94.24** $^{94.33}_{94.14}$ ±0.05	**95.98** $^{96.04}_{95.92}$ ±0.04	**94.24** $^{94.29}_{\mathbf{94.18}}$ ±0.03
6	**95.95** $^{95.98}_{\mathbf{95.91}}$ ±0.02	**94.24** $^{94.30}_{\mathbf{94.21}}$ ±0.03	**95.98** $^{96.01}_{\mathbf{95.94}}$ ±0.02	**94.24** $^{94.28}_{\mathbf{94.18}}$ ±0.03
R.E.R.	2.88%	3.03%	2.66%	2.87%

Table 4.1: PTB dev results, with First-Order (FOP) and Second-Order (SOP) parsers as experts.

4. `https://github.com/kidlestar/MOE.git`.

5. For English we used Glove embeddings [Pennington et al., 2014], while for Chinese we extracted pretrained embeddings from the publicly available model of [Zhang et al., 2020b].

K	BFOP		BSOP	
	UAS	LAS	UAS	LAS
1	$96.31^{96.46}_{96.23}$ ±0.06	$94.60^{94.77}_{94.53}$ ±0.06	$96.35^{96.42}_{96.23}$ ±0.04	$94.63^{94.68}_{94.55}$ ±0.04
2	$96.37^{96.48}_{96.26}$ ±0.05	$94.69^{94.79}_{94.61}$ ±0.05	$96.38^{96.51}_{96.26}$ ±0.06	$94.71^{94.79}_{95.60}$ ±0.05
3	$96.40^{96.49}_{96.33}$ ±0.04	$94.74^{94.82}_{94.68}$ ±0.04	$96.39^{96.50}_{96.29}$ ±0.06	$94.71^{94.79}_{94.61}$ ±0.04
4	$96.43^{\mathbf{96.53}}_{96.38}$ ±0.04	$94.77^{\mathbf{94.89}}_{94.72}$ ±0.04	$96.38^{96.47}_{96.28}$ ±0.05	$94.72^{94.79}_{94.62}$ ±0.05
5	$\mathbf{96.45}^{96.51}_{96.38}$ ±0.04	$\mathbf{94.79}^{94.85}_{\mathbf{94.74}}$ ±0.04	$\mathbf{96.41}^{\mathbf{96.52}}_{96.29}$ ±0.06	$\mathbf{94.73}^{\mathbf{94.82}}_{94.65}$ ±0.05
6	$96.44^{96.51}_{\mathbf{96.40}}$ ±0.03	$\mathbf{94.79}^{94.85}_{\mathbf{94.74}}$ ±0.03	$96.39^{96.46}_{\mathbf{96.32}}$ ±0.04	$\mathbf{94.73}^{\mathbf{94.82}}_{\mathbf{94.67}}$ ±0.04
R.E.R.	3.52%	3.52%	1.64%	1.86%

Table 4.2: PTB dev results, with Bert-First-Order (BFOP) and Bert-Second-Order (BSOP) parsers as experts.

4.6.1 Averaging Effect Analysis

The experimental procedure is shown in Experimental Setup 5, with $M_1, \ldots, M_{12}$ denoting the trained experts, K number of experts in the mixture model and r the number of repetitions.

Experimental Setup 5: Averaging Effect

Models: $M_1, \ldots, M_{12}$;
Initialization: K, r;
repeat r **times**
    1. Shuffle the order of $M_1, \ldots, M_{12}$;
    2. Combine sequentially every K models together, creating $12/K$ mixture averaging models;
    3. Compute UAS, LAS of models;
    4. Calculate system variance for models;
end

We set K from 1 to 6 with r always set to 5. We show results for PTB and CoNLL09 Chinese on dev data for each type of mixture of experts, and different number of experts in Table 4.1 and Table 4.3. For UAS and LAS, each entry is given as:

$$\text{Average}^{\text{max}}_{\text{min}} \pm \text{std}$$

where average is the *average* score for all trials in this setting and *max* (resp. *min*) is the highest (resp. *lowest*) score obtained by an experiment in this setting. We also give standard deviation *std* as a way to see the effects of variance reduction.

Finally the last row gives the average relative error reduction (R.E.R) from single expert mode ($K = 1$) to ensemble mode with $K = 6$.

K	FOP		SOP	
	UAS	LAS	UAS	LAS
1	$89.20\,^{89.42}_{89.04}\pm0.12$	$86.28\,^{86.49}_{86.10}\pm0.12$	$89.40\,^{89.48}_{89.31}\pm0.06$	$86.45\,^{86.52}_{86.28}\pm0.07$
2	$89.44\,^{89.60}_{89.32}\pm0.08$	$86.59\,^{86.74}_{86.45}\pm0.08$	$89.65\,^{89.78}_{89.51}\pm0.07$	$86.76\,^{86.89}_{86.57}\pm0.07$
3	$89.55\,^{89.66}_{89.39}\pm0.07$	$86.71\,^{86.82}_{86.54}\pm0.07$	$89.74\,^{89.86}_{89.62}\pm0.07$	$86.86\,^{86.98}_{86.72}\pm0.08$
4	$89.62\,^{89.68}_{89.52}\pm0.04$	$86.80\,^{86.87}_{86.70}\pm0.05$	$89.83\,^{89.94}_{89.70}\pm0.08$	$86.96\,^{87.08}_{86.86}\pm0.07$
5	$\mathbf{89.66}\,^{89.71}_{89.57}\pm0.04$	$86.83\,^{86.89}_{86.75}\pm0.04$	$\mathbf{89.87}\,^{\mathbf{89.98}}_{89.75}\pm0.07$	$\mathbf{87.00}\,^{\mathbf{87.11}}_{86.87}\pm0.07$
6	$\mathbf{89.66}\,^{\mathbf{89.77}}_{\mathbf{89.61}}\pm0.05$	$\mathbf{86.85}\,^{\mathbf{86.93}}_{\mathbf{86.81}}\pm0.04$	$\mathbf{89.87}\,^{89.93}_{\mathbf{89.79}}\pm0.05$	$\mathbf{87.00}\,^{87.08}_{\mathbf{86.92}}\pm0.04$
R.E.R.	4.26%	4.15%	4.43%	4.06%

Table 4.3: CoNLL09 dev results, with First-Order (FOP) and Second-Order (SOP) parsers as experts.

4.6.2 Clustering Effect Analysis

We conduct clustering effect analysis over the mixture model with 6 experts. Preliminary experiments showed that, like in most non-convex problems, good initialization is very important. For that reason we use already trained experts as starting points[6] although the mixture could benefit from more diversely trained experts. We leave this for future work. The procedure is described in Experimental Setup 6 and this whole procedure is repeated 5 times to compute average performance.

Experimental Setup 6: Clustering Effect

Models: $M_1, \ldots M_{12}$;
Initialization: $K = 6$;
repeat r **times**
    1. Select randomly K models, creating mixture models;
    2. Do fine tuning of mixture models with gating network;
    3. Calculating UAS, LAS of mixture model after fine tuning;
end

Scores on development set before and after fine tuning are shown in Table 4.4. Note that because shuffling might give different candidate sets than in the averaging experiments UAS and LAS results are not exactly the same as $K = 6$ results in Table 4.1, Table 4.2 and Table 4.3.

6. We tried deterministic annealing with both randomly initialized experts and already trained experts. While it helped in the former case, the latter was more accurate, but still less accurate than systems trained without.

Method	PTB		CoNLL09 Chinese	
	UAS	LAS	UAS	LAS
FOP	$95.94^{95.96}_{95.91}$ ±0.02	$94.23^{94.26}_{94.21}$ ±0.02	$89.67^{89.71}_{89.62}$ ±0.03	$\mathbf{86.86}^{86.91}_{86.81}$ ±0.04
CFOP	$\mathbf{95.98}^{96.00}_{95.94}$ ±0.02	$\mathbf{94.29}^{94.31}_{94.27}$ ±0.02	$\mathbf{89.68}^{89.72}_{89.62}$ ±0.04	$\mathbf{86.86}^{86.90}_{86.80}$ ±0.04
SOP	$95.98^{96.00}_{95.95}$ ±0.02	$94.23^{94.28}_{94.20}$ ±0.03	$89.85^{89.92}_{89.81}$ ±0.04	$86.98^{87.06}_{86.93}$ ±0.06
CSOP	$\mathbf{95.99}^{96.01}_{95.97}$ ±0.01	$\mathbf{94.25}^{94.28}_{94.22}$ ±0.02	$\mathbf{89.89}^{89.95}_{89.82}$ ±0.04	$\mathbf{87.03}^{87.12}_{86.95}$ ±0.06
BFOP	$\mathbf{96.43}^{96.46}_{96.42}$ ±0.02	$\mathbf{94.79}^{94.82}_{94.76}$ ±0.02	-	-
CBFOP	$96.42^{96.46}_{96.39}$ ±0.03	$94.78^{94.81}_{94.72}$ ±0.03	-	-
BSOP	$96.41^{96.46}_{96.37}$ ±0.04	$94.75^{94.82}_{94.67}$ ±0.05	-	-
CBSOP	$\mathbf{96.42}^{96.46}_{96.38}$ ±0.04	$\mathbf{94.76}^{94.82}_{94.70}$ ±0.05	-	-

Table 4.4: Clustering Effect with $K = 6$ on dev, where CFOP, CSOP, CBSOP represent models after training

4.6.3 Discussion

Averaging Tables 4.1 to 4.3 show that UAS and LAS generally increase on average with the number of models in the mixture model, and that ensemble performs often on average better than the best single systems in each category (notable exceptions: UAS for FOP and models with BERT on PTB).

Averaging generally decreases the standard deviation, which is evident for (B)FOP. For (B)SOP the decrease trend is less clear. However, we still found that the smallest standard deviation is usually given by high number of experts ($K = 5, 6$).

We remark that on PTB similar performance on dev was achieved by FOP and SOP, with a slightly better UAS for SOP, which is expected by the capacity of the model to better represent structures. This corroborates findings of [Falenska and Kuhn, 2019]. But this contradicts results for CoNLL09 where SOP always gives best results, in line with observations of [Fonseca and Martins, 2020]. For BERT experiments on PTB, BSOP achieves better performance than BFOP with one or two experts. However, when the number of experts increases, BFOP outperforms BSOP.

We complement our discussion with Figure 4.1[7] which depicts variance reduction by the number of experts in log-scale: almost linear of for all models, as predicted by our independence assumption.

We note that UAS and LAS improves little or not at all from $K = 5$ to $K = 6$. This is in accordance with the variance analysis for that the decrease of variance will become smaller when number of experts becomes higher. Indeed, applying Eq. (4.4), the decrease of variance from $K = 1$ to $K = 2$ is $\frac{1}{2}\sigma^2_{(h,d),x}$, while from $K = 5$ to $K = 6$ it is only $\frac{1}{30}\sigma^2_{(h,d),x}$, 15 times lower. This correponds to the observation the improvements of UAS

7. For CoNLL09, we found similar results. The figure is not shown for space limitation.

and LAS tend to decrease with the number of experts until it reaches a plateau.

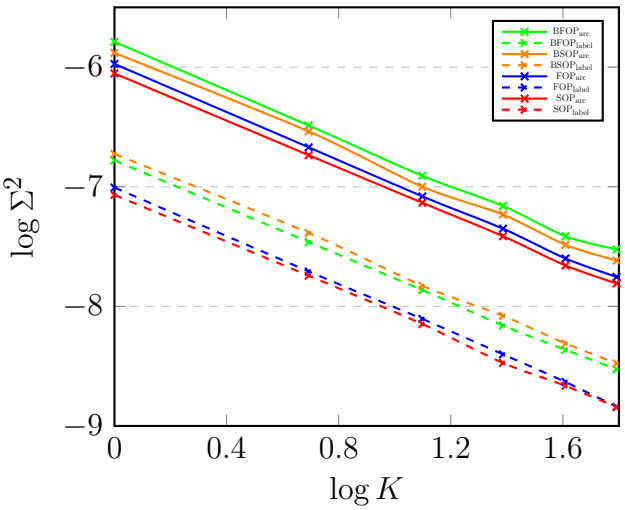

Figure 4.1: Variance by experts on PTB Dev Data.

Clustering We found that a modest improvement on UAS and LAS (0.01%-0.06% absolute) can be achieved by clustering (except for FOP on CoNLL09 Chinese). The average performance benefits generally from clustering while a tiny decrease (0.01%) is observed for BFOP on PTB.

Since FOP, SOP, BFOP and BSOP are all strong learners for PTB and CoNLL09 Chinese, *i.e.* UAS and LAS approaches 99% for both PTB and CoNLL09 on training data for all models, we can assume that an expert belonging to one of these models can learn efficiently most of the training data, as opposed to just a portion of it. Thus, only a a few of training instances can significantly be better covered by clustering. Moreover, as averaging has already achieved a considerable improvement (around 0.2%-0.6% absolute), a biased ω_k obtained from clustering may harm the gain from averaging.

4.6.4 Results on Test

Tables 4.5 and 4.6 show test results on PTB and CoNLL09, comparisons with recent models. We show test results of SOP and CSOP with 6 experts for PTB and CoNLL09. Additionally for PTB, we show BFOP, CBFOP, BSOP and CBSOP with 6 experts to make comparison with recent parsers, often more sophisticated than our approach, with BERT. We give the results with the same typographical system as [Zhang et al., 2020a] Please note that, while average results keep the same semantics, *max* and *min* give test results of the LAS highest-

and lowest- (resp.) scoring systems *on the development set*. We note that results of [Zhang et al., 2020a] would correspond our model with $K = 1$.

Method	PTB		CoNLL09 Chinese	
	UAS	LAS	UAS	LAS
[Dozat and Manning, 2017]	95.74	94.08	88.90	85.38
[Li et al., 2019]	95.93	94.19	88.77	85.58
[Ji et al., 2019]	95.97	94.31	-	-
[Zhang et al., 2020a]	96.14	94.49	89.63	86.52
FOP,$K = 6$	$96.20\,^{96.19}_{96.20}\pm0.02$	$94.64\,^{94.63}_{94.64}\pm0.02$	$89.91\,^{89.84}_{89.99}\pm0.06$	$87.00\,^{86.94}_{87.09}\pm0.07$
CFOP,$K = 6$	$96.20\,^{96.18}_{96.18}\pm0.02$	$94.65\,^{94.62}_{94.63}\pm0.02$	$89.94\,^{89.92}_{89.93}\pm0.04$	$87.03\,^{87.02}_{87.00}\pm0.04$
SOP,$K = 6$	$\mathbf{96.29}\,^{96.30}_{96.29}\pm0.02$	$\mathbf{94.71}\,^{94.72}_{94.73}\pm0.02$	$90.06\,^{90.14}_{89.97}\pm0.07$	$\mathbf{87.12}\,^{87.19}_{87.00}\pm0.07$
CSOP,$K = 6$	$96.27\,^{96.27}_{96.32}\pm0.03$	$94.69\,^{94.70}_{94.72}\pm0.03$	$\mathbf{90.07}\,^{90.00}_{89.99}\pm0.08$	$\mathbf{87.12}\,^{87.24}_{87.02}\pm0.09$

Table 4.5: Comparison on test sets without BERT.

Method	PTB	
	UAS	LAS
[Li et al., 2020]	96.44	94.63
[Mohammadshahi and Henderson, 2021]	**96.66**	95.01
BFOP,$K = 6$	$96.58\,^{96.60}_{96.57}\pm0.02$	$95.06\,^{95.07}_{95.02}\pm0.02$
CBFOP,$K = 6$	$96.58\,^{96.59}_{96.54}\pm0.02$	$95.06\,^{95.07}_{95.02}\pm0.02$
BSOP,$K = 6$	$96.64\,^{\mathbf{96.66}}_{96.58}\pm0.02$	$\mathbf{95.09}\,^{95.11}_{95.12}\pm0.03$
CBSOP,$K = 6$	$96.62\,^{\mathbf{96.66}}_{96.64}\pm0.03$	$95.07\,^{\mathbf{95.12}}_{95.07}\pm0.03$

Table 4.6: Comparison of BERT models on PTB test set.

For averaging models, we apply significance t-tests [Dror et al., 2018] with level $\alpha = 0.05$ to FOP, BFOP, SOP, BSOP with $K = 6$ against $K = 1$. For PTB and CoNLL09, p-value is always smaller than 0.005. We note that for parsers without BERT, averaging can achieve a considerable improvement with SOP and gives new SOTA. We also point out that, if FOP and SOP could find equivalently good models on dev, SOP models seem to better generalize. For parsers with BERT, with a simple averaging of BSOP, we achieve comparable performances (or even better in case of LAS) when comparing to more involved methods such as [Li et al., 2020, Mohammadshahi and Henderson, 2021]. It remains to be seen whether they can also benefit from MoEs.

Regarding clustering, even if we obtained an average improvement on dev, test data hardly benefits from it. Still, we note a small improvement of UAS on SOP CoNLL09. Finally we stress that best performing settings on PTB test, namely BSOP and CBSOP, were not better performing than BFOP and CBFOP on development data on average (although max systems were similar): second-order models seem to slightly better handle unseen data.

4.6.5 Parallel Training and Decoding

Training averaging ensembles can be paralleled with sufficient GPUs, since each expert is trained independently. For fine tuning with clustering, most of the training could in principle be paralleled as well, although for the sake of simplicity we didn't implement such a training procedure: the training time of clustering model increases linearly with number of experts. As we only need a few epochs for fine tuning, the overall training time is comparable to training a single expert.

For decoding, calculations are performed in parallel as well. First marginal probabilities for arcs and labels are computed for every expert in parallel. Then they are combined either as a simple average or as a weighted sum. Finally, we apply the decoding algorithm (LMBR) *once* over the combined probability. The overhead is thus quite limited, for instance with $K = 6$ the overall decoding time is only around 10% higher than with a single expert.

4.7 Related Work

Ensembling parsers showed good results in shared tasks Che et al. [2018][8] and were framed as a combination of experts in Petrov [2010]. In this chapter we show how this is related to mixtures and distinguish averaging and clustering effects.

The use of mixture model for syntactic parsing was introduced in Petrov et al. [2006] for PCFG models, where it provided an access to non-local features unreachable to mere PCFGs. However, now that powerful non-Markovian feature extractors (*i.e.* BiLSTMs or Transformers) are widely used, the expected gain is more difficult to characterize, but we hypothesize that it is related to the softmax bottleneck Yang et al. [2018] implied by using different exponential models in all predictions, even when richly parameterized.

We modelled parser combinations with finite mixture models, but more sophisticated parsing models Kim et al. [2019] use infinite mixture models. In this case it might be more difficult to discriminate between averaging and clustering. Our mixture is essentially a latent variable model where the latent variables range over experts. Although inspired from EM with neural networks, similarly to Nishida and Nakayama [2020], other methods based on ELBo and sampling could also be utilized Corro and Titov [2019], Zhu et al. [2020].

8. Ensembling is widely used in Machine Translation shared tasks, such as WMT.

4.8 Conclusion

We framed dependency parser combination as a finite mixture model, showed that this model presents two distinct properties –an averaging effect and a clustering effect– and devised an efficient decoding method. Moreover, we studied the impact of the averaging effect, namely variance reduction during training, and consequently better accuracy. We investigated the reasons of instability when learning mixture models, and proposed an EM-inspired method to avoid over-specialization. When used as fine-tuning, this method may improve accuracy over averaging. As a by-product, this method gives state-of-the-art results when combined with first-order and second-order projective parsers on two standard datasets.

This work can be further expanded in future research: the increase of parameters can be seen as overparameterization, and many parameters must be redundant. A potentially fruitful avenue of research could be the investigation of the subnetwork hypothesis, *i.e.* whether distillation could give a smaller network with similar performance. Moreover, the use of different types of experts for MoE can be explored, which may benefit the clustering effect.

CHAPTER 5
POLYNOMIAL MODEL FOR DEPENDENCY PARSING

5.1 Introduction

Score functions of graph-based models are limited as constrained polynomial functions to ensure the existence of efficient inference algorithms (see Chapter 3 for details). As far as we know, deep higher-order models use only some of the second-order structures (siblings and grandchild). In this chapter, we propose a *generalized* polynomial model, which can incorporate all possible structures of any order.

The goal of modern graph-based dependency parsing is to find the most adequate parse structure for the given input sentence by computing a score for all possible candidate parses, and returning the highest-scoring one. Since the number of candidates is exponential in the sentence length, the scoring is performed implicitly: after computing scores for possible parts, the best structure, whose score is the sum of its various parts, is returned by a combinatorial algorithm based on either dynamic programming such as the Eisner algorithm [Eisner, 1997] in the projective case, or duality gap such as the Chu-Liu-Edmonds algorithm [McDonald et al., 2005] in the non-projective case.

Graph-based models where parts are restricted to single arcs are called first-order models, while models where parts contains k-tuples of arcs are called k^{th}-order models. For instance models with score for sibling and grand-parent relations are 2^{nd}-order models because parts consist of 2 *connected* arcs. The connectivity is important since it helps building efficient dynamic programming algorithms in the case of projective arborescences [Koo and Collins, 2010] or efficient approximations in the non-projective case based on lagrangian heuristics [Koo et al., 2010, Martins et al., 2013] or belief propagation [Smith and Eisner, 2008]. The score function of first-order models, being a sum of parts which are simple arcs, is linear in arc variables, while for second-order, being a sum of parts which are pair of arcs, the score function is quadratic in arc variables. More generally k^{th}-order models have a polynomial score function in arc variables, with highest degree equal to k.

In this chapter we explore the consequences of treating score functions for higher-order dependency parsing as polynomial functions. This framework can recover most previously defined score functions and gives a unified framework for graph-based parsing. Moreover, it can express novel functions since in this setting parts are made of possibly disconnected tuples of arcs. We call the results *generalized* higher-order models, as opposed to previously *connected* higher-order models.

On the other hand, polynomial functions are difficult to manipulate. They are non-convex and so, in addition to already known problems in higher-order parsing such as the

computation of the partition function for probabilitic models, MAP decoding is itself a challenge. We develop an approximate parsing strategy based on coordinate ascent [Bertsekas, 1999], where we iteratively improve a candidate by flipping arcs. We exploit the polynomial nature of the score function to derive an accurate and efficient procedure to select arcs to be flipped. Since coordinate ascent converges to a local minimum, we show how this method can be embedded within a meta-heuristics based on genetic analogy [Schmitt, 2001] to find better optima.

We can learn these models via two methods, max-margin or probabilitic estimation. Max-margin is straightforward because it only requires MAP decoding but is quite fragile since it is sensitive to approximation errors which are inevitable in our setting. We design a probabilistic loss for our model where we approximate parse scores via a first-order Taylor expansion around the MAP solution. We find that this novel method is efficient and we show empirically that it can outperform previous higher-order models.

In summary our contributions are the following:

- a general framework for dependency parsing which encompasses previous higher-order score functions, and includes new ones;
- a new method for higher-order dependency parsing based on non-linear optimization techniques (coordinate ascent and genetic algorithm) coupling gradient-based methods, and combinatorial routines;
- an empirical validation of this method which obtains state-of-the-art results on standard datasets and is computationally efficient.

5.2 Related Work

Before the use of powerful neural feature extractors (*e.g.* BiLSTM or Transformers) dependency parsing with high-order relations was a clear improvement over first-order models. [Koo and Collins, 2010] considered efficient third order models for projective dependency parsing. In order to have efficient dynamic programming algorithms for decoding, only a few limited predefined structures can be included to the model (*e.g.* dependency, sibling, grandchild, grand-sibling, tri-sibling).[1]

Higher-order non-projective parsing is NP-hard but fast heuristics with good performance have been proposed based on dual decomposition for instance. However, efficient subsystems

1. The term *sibling* often means *adjacent sibling*, where only adjacent modifiers on the same side of the head are included.

must be devised to efficiently process complex parts, either based on dynamic programming algorithms such as Viterbi [Koo et al., 2010] or on integer linear programming [Martins et al., 2013]. In practice this restricts parts to connected subgraphs.[2]

Since the wide adoption of deep feature extractors, the situation is less clear. [Zhang et al., 2020a] consider a second-order model with dependency and adjacent sibling, which can guarantee efficient decoding for projective arborescence with a batchified variant of Eisner algorithm [Eisner, 1996, 1997]. The results show that adjacent sibling is beneficial for the performance of parser comparing with arc-factored model. [Fonseca and Martins, 2020] claim that in the non-projective case, second-order features help especially in long sentences. On the other hand, [Falenska and Kuhn, 2019] showed that in general the impact of consecutive sibling features was not substantial, and [Zhang et al., 2021] showed that the main benefit of these features could be understood as variance reduction, and vanishes when ensembles are used.

Closely related to our work, [Wang and Tu, 2020] consider a second-order model with score for dependencies, siblings and grandchildren where they do not constrain siblings to be adjacent. Although exact estimation is intractable in their setting, an approximate estimation of probability of arborescences can be calculated efficiently by a message-passing algorithm. Their experiments seem to confirm that second-order relations are beneficial to the parsing accuracy, even when trained by an approximate estimation of probability, namely Mean-Field Variational Inference. Instead we approximate the partition function using a first-order Taylor approximation around the solution of the MAP solution. Partition approximations are usually performed via Bethe's free energy, see for instance [Martins et al., 2010, Wiseman and Kim, 2019].

[Dozat and Manning, 2017] showed that head selection was a good trade-off during the learning phase, for first-order models. Our method applies this principle to the higher-order case, leading to a coordinate ascent method, well known in the optimization literature [Bertsekas, 1999]. In Machine Learning and NLP, ascent methods are usually performed in primal-dual algorithms, *e.g.* [Shalev-Shwartz and Zhang, 2013] for SVMs.

We use genetic programming to escape local optima when searching for the best parse. Although this kind of metaheuristics has been used for other tasks in NLP such as WSD [Decadt et al., 2004] or summarization [Litvak et al., 2010], it is the first time it is applied to dependency parsing to the best of our knowledge. Since genetic algorithms can be seen as implementing a Markov-Chain [Schmitt, 2001] over candidate solutions, our method resembles MCMC methods, related to Gibbs sampling for Metropolis-Hastings methods, which

2. We note that [Martins et al., 2013] used a 2-arc part called *adjacent modifiers* which is not a connected subgraph. But this was not generalized to 2-arc arbitrary subgraphs.

have already been investigated in parsing [Zhang et al., 2014, Gao and Gormley, 2020]. Our method to choose the best arc to improve the current parse is inspired by a recent method for sampling in discrete distributions [Grathwohl et al., 2021] where we replace sampling by MAP inference.

We rely on properties of polynomials to derive efficient routines for approximate head selection. Polynomial factors where discussed for higher-order parsing in [Qian and Liu, 2013].

5.3 Notations

We note C_x as the set of all possible arcs for sentence x (the arcs of the complete graph over vertices in x) or C when unambiguous.

We say that a non-empty set of arcs $A = \{(h_1, d_1), \dots (h_k, d_k)\}$ is *proper* if $\forall i, h_i \neq d_i$ and $\forall i < j, d_i \neq d_j$. The first condition asserts that an arc cannot be a self-loop while the second enforces that each word has only one head in a proper set. The two constraints are natural and required for dependency parsing. We note the set of proper subsets of cardinal k which can be constructed from a set of arcs A as $\mathcal{F}_k(A)$, the set of k^{th}-order polynomial factors.

5.4 Polynomial Score Functions for Dependency Parsing

In this work, we consider a generalization of previously proposed score functions for graph-based dependency parsing. Unlike higher-order models which consider only limited higher-order relations, *e.g.* Koo and Collins [2010], the proposed function can express all possible higher-order relations and can be viewed as a natural generalization of Wang and Tu [2020], Zhang et al. [2020a].

5.4.1 Score Function

We define K^{th}-order score functions as:

$$\begin{aligned} S(x, y) &= \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \prod_{(h,d) \in F}^{k} y_{hd} \\ &= \sum_{k=1} \sum_{F \in (\mathcal{F}_k(y) \cap \mathcal{R})} s_F \end{aligned} \tag{5.1}$$

where s_F represents the score for the higher-order factor constructed from arcs in F, and $\mathcal{R}$ is set of authorized structures (the *restriction*). Remark that Eq. (5.1) does not enforce a specific structure for $y \in \mathcal{Y}$ and could be applied in $\mathcal{G}, \mathcal{A}, \mathcal{P}$.

With this general definition we can recover most previous models for graph-based dependency parsing. For instance, in [Wang and Tu, 2020], a second order model ($K = 2$) is studied where only sibling and grandchild relations are considered, which can be expressed with the following $\mathcal{R}$: for $F = \{(h_1, d_1), (h_2, d_2)\}$, we enforce $h_1 = h_2$ or $d_1 = h_2$. In Zhang et al. [2020a], another second-order model, the restriction is stricter in order to limit acceptation to adjacent siblings: $h_1 = h_2$ and $(h_1, d_1), (h_2, d_2)$ are adjacent (no arcs from h_1, h_2 to word between d_1, d_2).

To demonstrate the generality of this approach, we also consider a generalized third-order model. The first-order and the second-order parts are as [Wang and Tu, 2020], and for third-order factors $F = \{(h_1, d_1), (h_2, d_2), (h_3, d_3)\}$, we add restrictions $d_1 < d_2 < d_1 + 3$ and $d_2 < d_3 < d_2 + 3$. Arcs in F are not always connected. Instead, we only force the modifiers of arcs to be close, with a maximum distance set to 2. This addition of cubic factors could be a computational bottleneck since it would naively require computing $O(n^6)$ scores. We avoid this with tensor factorization following [Peng et al., 2017].[3]

5.4.2 Score of One-Arc Modifications

Parsing can be framed as finding the highest $S(x, y)$, or $S(y)$ when x is unambiguous:

$$y^* = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, S(y) \tag{5.2}$$

The solution is tractable for $K = 1$ (first-order model), *i.e.* arc-factored model, for all usual parse structures, such as $\mathcal{G}, \mathcal{A}, \mathcal{P}$. However, it is intractable without additional constraints for higher-order models, such as projectivity for parses and adjacent siblings in scores.

We consider here a simpler problem: how much can the score change if we change **one** arc of the current parse? The idea is that better parses may be obtained by choosing arcs to be flipped. Thus, even starting with a *bad* parse, we may approach the *best* parse by modifying one arc at a time.

To solve this simpler problem, the naive method, *i.e.* calculate the score of every parse which differs from the current parse by one arc, is obviously inefficient and unpractical since it requires $O(n^2)$ evaluations (for each modifier and each head). Instead, we show that the score change of a one-arc modification can be efficiently calculated for Eq. (5.1). Let us

3. See Appendix C.3 for details.

consider the current parse y and an arbitrary arc $a = (h, d) \in C$ (possibly not in y). The partial derivative of the score to variable y_a is:[4]

$$\begin{aligned}\frac{\partial S(y)}{\partial y_a} &= \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a} \\ &= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ a \in F}} s_F \mathbf{1}[F \backslash a \in \mathcal{F}_{k-1}(y)]\end{aligned} \tag{5.3}$$

We can interpret this formula for the partial derivative as the sum of all factors F including a which verify $(F \backslash a) \in \mathcal{F}_{k-1}(y)$. When $a \in y$, $\frac{\partial S(y)}{\partial y_a}$ can be seen as the restriction of $S(y)$ to factors $F \subseteq \mathcal{F}_k(y)$, where $a \in F$. And we can write:[5]

$$S(y) = \frac{\partial S(y)}{\partial y_a} + S(y \backslash a) \tag{5.4}$$

where the last term is the score of all factors in y which do not contain a.

When $a \notin y$, we note $a = (h', d)$ while we assume $(h, d) \in y$.We define $y[h \to h', d]$ as the parse which modifies y by swapping the head index for column d from h to h' while the other columns remain unchanged, and $y[\to h', d]$ when the current head h is unimportant. We can rewrite the score function of $y[h \to h', d]$ with the previously defined partial derivative, and take advantage of the score factorisation to express $S(y[h \to h', d])$ with quantities directly available on y:[6]

$$S(y[h \to h', d]) = \frac{\partial S(y)}{\partial y_{h'd}} + S(y \backslash (h, d)) \tag{5.5}$$

Remark that the right part of the equation concerns only the original arborescence y.

We write $D(y \to h', d)$ for the change of score induced by swapping the head in column d to h', and $D(y, h \to h', d)$ when we want to emphasize that the current head for d is h. From the previous equations, we can derive:

4. See Appendix C.2.1 for the detailed derivation.

5. See Appendix C.2.2 for the detailed derivation.

6. See Appendix C.2.3 for the detailed derivation.

$$
\begin{aligned}
D(y, h \to h', d) &= S(y[(h \to h', d)]) - S(y) \\
&= \frac{\partial S(y)}{\partial y_{h'd}} - \frac{\partial S(y)}{\partial y_{hd}}
\end{aligned} \tag{5.6}
$$

Thus, to have a complete evaluation of change of scores, we only need one forward and backward evaluation on the score of the current solution y and then compute differences for each position d. In the following section, we build an inference algorithm based on this observation

5.5 Inference as Candidate Improvement

5.5.1 Coordinate Ascent

The main idea of our method is, from an initial parse y_0, to change the current candidate by picking a word and swapping its head to improve the score function. This is repeated until not further improvement is possible. This method is an instance of coordinate ascent Bertsekas [1999] (Chap. 2.7), to maximize Eq. (5.1). When parses are arborescences, such as when working in $\mathcal{A}$ and $\mathcal{P}$, this method must at each step, not only pick an improving arc, but also assert that the resulting parse has the required tree structure. This adds complexity that we propose to avoid by working in $\mathcal{G}$ and inserting a final step of projection to recover a solution in the desired space (described in Section 5.6.2).

Remark that when arborescence constraints are dropped, finding the best parse reduces to head selection, *i.e.* choose $h_d, \forall d$ with $y_{h_d,d} = 1$, which maximizes $S(y)$. To emphasize that this method works *column by column* we write:

$$
S(x, y) = S(y_{:,1}, \ldots, y_{:,|x|})
$$

where $y_{:,d}$ denotes the one-hot vector where $y_{:,d}[h] = 1$ if $(h, d) \in y$.

This is straightforward and tractable for first-order models, since it amounts to maximizing independent score functions.

However, this becomes intractable in higher-order models since parts overlap. Still, a local optimum can be obtained by coordinate ascent.

Given a current solution y^k, basic coordinate ascent finds a better next iterate y^{k+1} by cycling through columns and improving the current solution locally by successive head selections:

$$h_d^* = \underset{h}{\operatorname{argmax}}\, S(y_{:,1}^{k+1}, \ldots, y_{:,d-1}^{k+1}, \xi_h, y_{:,d+1}^{k} \ldots, y_{:,|x|}^{k}) \tag{5.7}$$

where ξ_h is the one-hot vector with 1 at position h. We set $y_{:,i}^{k+1} = \xi_{h_d^*}$ and the process is repeated for every word until there is no change ($y^{k+1} = y^k$).

5.5.2 A Gradient-based Method For Coordinate Ascent

A naive method to solve Eq. (5.7) requires n evaluations of S, one per possible head, which is inefficient. However, from Section 5.4.2 and Eq. (5.6), we can rewrite Eq. (5.7) since it amounts to finding a better head at position d from current solution y:

$$h_d^* = \underset{h}{\operatorname{argmax}}\, D(y \rightarrow h, d) \tag{5.8}$$

Thus, one forward and one backward (followed by $|x|$ substractions) is sufficient to decide the modification of arc at each position d.

Still, the gradient-based maximization presented above requires n forward and backward passes to determine the new heads for all words of the sentence. In order to achieve faster convergence, we want to avoid cycling through each word and consider the following problem: at each step, find the pair (h, d) which provides the greatest positive change in the score function:

$$(h^*, d^*) = \underset{h,d}{\operatorname{argmax}}\, S(y_{:,1}^{k}, \ldots, y_{:,d-1}^{k}, \xi_h, y_{:,d+1}^{k} \ldots, y_{:,|x|}^{k}) \tag{5.9}$$

We set $y^{k+1} = y^k[\rightarrow h^*, d^*]$ while other columns are unchanged. This is repeated until $y^{k+1} = y^k$.

Again, a naive maximization requires $O(n^2)$ estimations of score for each step and brings in fact no speed gain. However, as we have already seen, Eq. (5.9) is simply equivalent to:

$$(h^*, d^*) = \underset{h,d}{\operatorname{argmax}}\, D(y, \rightarrow h, d) \tag{5.10}$$

which again requires one forward and backward on the current candidate's score before substractions.

In summary our algorithm, from an initial parse y_0, iteratively improves a current solution: at step k we solve Eq. (5.10) by computing the gradient of $S(y^k)$ over arc variables and then pick the arc (h, d) whose partial derivative increases the greatest to set $y^{k+1} = y^k[\rightarrow h, d]$.

5.5.3 First-Order Linearization

Coordinate ascent changes one arc at a time which can still be slow. In practice, we found that a simpler greedy method performed at the beginning of the search, when high precision is not required, can improve parsing time dramatically. Given a current solution y^k, we linearize the score function via the first-order Taylor approximation and apply coordinate ascent to what is now an arc-factored model where columns can be processed independently. For each sentence position d:[7]

$$h_d^* = \operatorname*{argmax}_h \frac{\partial S(y^k)}{\partial y_{hd}^k}.$$

We set then $y_{:,d}^{k+1} = \xi_{h_d^*} \forall d > 0$. We may be able to change $|x|$ arcs at each step k, and the process is repeated until $S(y^{k+1}) \leq S(y^k)$, which indicates that the approximation becomes detrimental, after which we switch to coordinate ascent to provide more accurate iterations.

5.5.4 Genetic Algorithm

Due to the non-convexity of function S, the previous method gives a local optimum, which may limit the usefulness of higher-order parts. Thus, to ensure a better approximation, we add genetic search [Mitchell, 1998].

Genetic Algorithm is an evolutionary algorithm inspired by the process of natural selection. The algorithm requires: a solution domain, here $\mathcal{G}$, and a fitness function, *i.e.* function $S(y)$. Each step in our genetic algorithm consists of four consecutive processes: selection, crossover, mutation and self-evolution, which are repeated until stabilization.

Selection For a group of parses $y_1, \ldots, y_w$, estimate scores $S(y_1), \ldots, S(y_w)$. Select the top-k candidates ($k < w$) $y_1^s, \ldots, y_k^s$.

Crossover Average candidates $y^c = \frac{1}{k}\sum_{i=1}^{k} y_k^s$. Set $y_{h,d}^c$ as the probability of having (h, d) in an optimal parse and sample $w - k$ new parses according to y^c. Note them $y_1^c, \ldots, y_{w-k}^c$.

Mutation For every parse in $y_1^c, \ldots, y_{w-k}^c$, change heads randomly with probability p. Note mutated parses as $y_1^m, \ldots, y_{w-k}^m$

Self-Evolution On parses $y_1^m, \ldots, y_{w-k}^m$, apply coordinate ascent. Note the output as $y_1^e, \ldots, y_{w-k}^e$. Combine new parses with the previous top-k parses as the group for next iteration.

Selection and self-evolution pick arcs giving high scores while crossover and mutation can provide the possibility to jump out of local optima. We iterate this process until the best

7. See Appendix C.2.4 for the detailed derivation.

parse is unchanged for t consecutive iterations.

5.6 Learning and Decoding with Polynomial Scores

5.6.1 Learning

We follow recent works [Zhang et al., 2020a, Wang and Tu, 2020] and learn parse structures and arc labels in a multitask fashion with a shared feature extractor. Loss is the sum of label and arc losses:

$$L = L_{\text{label}} + L_{\text{arc}} \tag{5.11}$$

We write (x^*, y^*, l^*) as the training input sentence and its corresponding parse and labeling.

Label Loss Following [Dozat and Manning, 2017], we use the negative log-likelihood:

$$L_{\text{label}}(x^*, y^*, l^*) = -\sum_{(h,d)\in y^*} \log p(l^*_{hd}|x^*).$$

Hinge Loss Following [Kiperwasser and Goldberg, 2016], we can use hinge loss as arc loss:

$$L_{\text{arc}} = \text{ReLU}(\max_{y\in\mathcal{Y}} S(x^*, y) - S(x^*, y^*) + \Delta(y, y^*))$$

where $\Delta(y, y^*)$ is the Hamming distance.

The inner maximization requires to solve an inference sub-problem, *i.e.* to find the cost-augmented highest-scoring parse:

$$\max_{y\in\mathcal{Y}} S(x^*, y) + \Delta(y, y^*) \tag{5.12}$$

As Hamming distance is not differentiable, we propose to reformulate it as:

$$\Delta(y, y^*) = \sum_{h,d} (1 - y_{hd})y^*_{hd} + (1 - y^*_{hd})y_{hd}$$

linear to the variable y. Thus, Eq. (5.12) can be solved with the method proposed in Section 5.5.

Approximate Marginal Estimation In practice hinge loss may have two issues: each update is limited to two parses only, which makes learning slow, and the linear margin may

lead to insufficient learning. We thus propose an approximate probabilistic learning objective inspired by methods such as Mean-Field Variational Inference [Wang and Tu, 2020]. We would like to train our model as an arc-factored log-linear model:

$$L_{\text{arc}} = -\sum_{(h,d)\in y*} \log p\big((h,d)|x^*\big)$$

where $p\big((h,d)|x^*\big)$ is the marginal probability of arc (h,d) over all parses for x^*.

Marginal probabilities are approximated based on the intuition that the distribution of parses is usually peaked on few close solutions, hence that estimating the contribution of arcs at the neighborhood of the highest-scoring parse gives an acceptable approximation. We use the same reasoning as in Section 5.5.3 to derive a linear approximation of the current model. Given parse $\hat{y}$ obtained by coordinate ascent, we set:[8]

$$\begin{aligned} p\big((h,d)|x^*\big) &= \frac{p(\hat{y}[\rightarrow h,d])}{\sum_{h'} p(\hat{y}[\rightarrow h',d])} \\ &\approx \frac{\exp(s_{hd})}{\sum_{h'} \exp(s_{h'd})} \end{aligned} \tag{5.13}$$

where:

$$s_{hd} = \frac{\partial S(\hat{y})}{\partial y_{hd}} \tag{5.14}$$

5.6.2 Approximate MBR Structured Decoding

Inference with coordinate ascent and genetic algorithm cannot guarantee the tree structure of parses, as they work in solution space $\mathcal{G}$. But we can estimate the marginal probability of arcs from a solution y returned by coordinate ascent by reusing Eq. (5.13). Then, the Eisner algorithm [Eisner, 1996, 1997] or the Chu-Liu-Edmonds algorithm [McDonald et al., 2005] can be applied to have projective or non-projective arborescences. We remark that this is similar to Minimum Bayesian Risk (MBR) decoding [Smith and Smith, 2007a], the difference being that here marginalization is estimated with nearest arborescences while for MBR marginalization is exact over the parse forest.

8. See detailed derivation in Appendix C.2.5.

	bg	ca	cs	de	en	es	fr	it	nl	no	ro	ru	Avg.
CRF2O	90.77	91.29	91.54	80.46	87.32	90.86	87.96	91.91	88.62	91.02	86.90	93.33	89.33
Local2O	90.53	92.83	92.12	81.73	89.72	92.07	88.53	92.78	**90.19**	91.88	85.88	92.67	90.07
CA+LM	90.79	93.14	**91.92**	**84.45**	**89.89**	**92.60**	**90.14**	93.57	89.89	**93.85**	**86.42**	93.81	**90.87**
3O+CA+LM	**90.80**	93.09	91.91	84.42	89.75	92.50	90.02	93.53	90.13	93.78	86.38	93.86	90.85
GA+CA+LM	90.70	**93.17**	91.90	84.19	89.77	92.50	89.88	**93.68**	90.13	93.81	86.33	**93.88**	90.83
						+BERT							
Local2O	91.13	93.34	92.07	81.67	90.43	92.45	89.26	93.50	90.99	91.66	86.09	92.66	90.44
CA+LM	**91.93**	**94.09**	92.46	**85.59**	90.97	93.42	90.88	94.18	91.49	94.57	87.22	94.40	**91.77**
3O+CA+LM	91.87	94.05	**92.50**	85.22	91.04	**93.47**	90.79	**94.26**	91.38	**94.62**	87.18	94.41	91.73
GA+CA+LM	91.86	94.08	92.49	85.38	**90.99**	93.44	**91.05**	94.13	**91.53**	94.56	**87.25**	**94.42**	**91.77**

Table 5.1: LAS on UD 2.2 test data. CRF2O: [Zhang et al., 2020a]; Local2O: [Wang and Tu, 2020].

5.7 Experiments

We present experimental results[9] where we evaluate and compare our parsing method where we use the score function [Wang and Tu, 2020] and our extension with third-order factors (3O) with coordinate ascent (CA) and genetic algorithm (GA).

5.7.1 Data

Two datasets are used for projective dependency parsing: the English Penn Treebank (PTB) with Stanford Dependencies [Marcus et al., 1993] and CoNLL09 Chinese data [Hajič et al., 2009]. We use standard train/dev/test splits and evaluate with UAS/LAS metrics. Punctuation is ignored on PTB for dev and test. For non-projective dependency parsing, Universal Dependencies (UD) v2.2 is used. Following [Wang and Tu, 2020], punctuation is ignored for all languages.

For experiments with BERT [Devlin et al., 2019], we use BERT-Large-Uncased for PTB, BERT-Base-Chinese for CoNLL09 Chinese and Base-Multilingual-Cased for UD.

5.7.2 Hyper-Parameters

To ensure fair comparison, and for budget reasons, we use the same setup (hyper-parameters and pre-trained embeddings) as [Zhang et al., 2020a].[10]

For experiments without BERT [Devlin et al., 2019], pos-tags are used for all datasets[11].

9. Our prototype will be publicly available upon publication.

10. See Appendix C.1.

11. In [Zhang et al., 2020a], pos-tagging used on UD but not on PTB nor CoNLL09 Chinese. In [Wang and Tu, 2020], pos-tagging is used for all datasets.

Method	PTB		CoNLL09	
	UAS	LAS	UAS	LAS
CA+hinge	95.69	93.89	91.25	89.52
GA+CA+hinge	95.71	93.87	**91.52**	**89.80**
CA+LM	95.67	93.88	91.31	89.66
3O+CA+LM	95.64	93.87	91.26	89.61
GA+CA+LM	**95.81**	**93.99**	91.30	89.66
+BERT				
CA+LM	**96.53**	**94.85**	**93.18**	**91.57**
3O+CA+LM	96.47	94.79	93.15	91.53
GA+CA+LM	96.50	94.82	93.16	91.55

Table 5.2: Comparison on dev. CA: Coordinate Ascent; 3O: Third order model; GA: Genetic Algorithm; LM: Linearized Marginalization; hinge: hinge loss

For experiments with BERT, the last 4 layers are combined by scalarmix and linear projection and then concatenated to the original feature vectors.

Initial candidates are sampled from the the first-order part of Eq. (5.1). For genetic algorithm, due to hardware memory limitations, the number of candidates is set to 6. Each time, we take the top-3 candidates in selection, and the genetic loop is terminated when the best parse remains unchanged for 3 consecutive iterations. The mutation rate is set to 0.2 on all datasets.[12]

All experiments are run 3 times with random seed set to current time and averaged. We rerun also the results of [Wang and Tu, 2020] on PTB and CoNLL09 with the authors' implementation.[13] to have a faire comparison.

5.7.3 Results on PTB and CoNLL09 Chinese

Table 5.2 shows results of our different system with and without BERT. For PTB without BERT we see that training via coordinate ascent with hinge loss of marginal estimation give similar results, while genetic algorithm gives a sensible improvement when combined with the probabilistic framework. We can see that our third-order factors do not improve scores. With BERT probabilistic models, neither third-order nor genetic algorithm on top of coordinate

12. We tried mutation rates $0.1, 0.2, 0.3$ and the best performance is obtained on PTB dev with mutation rate 0.2.

13. `https://github.com/wangxinyu0922/Second_Order_Parsing`, Note that this implementation also uses the hyper-parameters of [Zhang et al., 2020a]

Method	PTB		CoNLL09	
	UAS	LAS	UAS	LAS
CRF2O*	**96.14**	**94.49**	89.63	86.52
Local2O	95.98	94.34	-	-
Local2O†	95.90	94.25	91.60	89.93
CA+hinge	95.88	94.21	91.27	89.58
GA+CA+hinge	95.93	94.26	91.63	89.89
CA+LM	95.96	94.33	91.62	89.96
3O+CA+LM	95.85	94.27	91.59	89.96
GA+CA+LM	95.95	94.34	**91.65**	**90.02**
+BERT				
Local2O	**96.91**	**95.34**	-	-
Local2O†	96.68	95.16	93.46	91.87
CA+LM	96.68	95.20	**93.48**	**91.91**
3O+CA+LM	96.65	95.13	93.47	91.87
GA+CA+LM	96.67	95.20	93.42	91.83

Table 5.3: Comparison on test. *: POS not used. †: Rerun with official implementation.

ascent gives any improvement. For CoNLL09 Chinese without BERT, performance on dev are similar between settings while genetic algorithm gives an evident boost for hinge loss. With BERT as for PTB the simple model performs best.

Table 5.3 gives test results and comparisons with two recent similiar systems. For PTB without BERT, the exact projective parser of [Zhang et al., 2020a] has the best performance, which is in accordance with the reported results in [Wang and Tu, 2020].[14] In comparison with [Wang and Tu, 2020] (Local2O), although their system has more parameters for PTB experiments[15], our coordinate ascent method with genetic algorithm plus marginalization has achieved the same performance on LAS. However, the same optimization method with hinge loss does not show good performances. For CoNLL09 Chinese without BERT, the genetic algorithm seems to help with generalization compared to simple coordinate ascent (similar score on dev but improvement on test).

With BERT, on both corpora, simple coordinate ascent gives best performance for our

14. Our best single run gives 94.44 LAS on PTB which is on a par with their results.

15. [Wang and Tu, 2020] uses a bilstm with hidden 600 while we follow [Zhang et al., 2020a] to use a bilstm with hidden size 400

method.

5.7.4 Results on UD

Table 5.1 shows LAS on UD test data. The best average performances are achieved with coordinate ascent and genetic algorithm plus linearized marginalization. For all languages, our method with or without genetic algorithm outperforms [Wang and Tu, 2020] (Local2O) except **nl** without BERT.

Method	Train	Test
Local2O	1133	706
CA	506	399
3O+CA	255	249
GA+CA	248	195

Table 5.4: Speed Comparison on PTB Train and Test without BERT (sentences per second)

5.7.5 Speed Comparison

We compare the speed of train and test with Nvidia Tesla V100 SXM2 16 Go on PTB. The result is shown in Table 5.4. For coordinate ascent, training is 2.2 times slower than MFVI while test is 1.8 times slower than MFVI[16].

5.8 Conclusion

We presented a novel method for higher-order parsing based on coordinate ascent. Our method relies on the general form of arc-polynomial score functions. Promising arcs are picked by evaluated by gradient computations. This method is agnostic to specific score functions and we showed how we can recover previously defined functions and design new ones. Experimentally we showed that, although this method returns local optima, it can obtain State-of-the-art results.

Further research could investigate whether the difference between the search space during learning and decoding is a cause of performance decrease. In particular the coordinate ascent could be replaced by a structured optimization method such as the Frank-Wolfe algorithm

16. The speed is measured with Eisner applied over all sentences. It is about 2 times quicker with the faster decoding strategy of [Zhang et al., 2020a] which consists in applying Eisner only if the coordinate ascent solution does not return a projective arborescence.

(see [Pedregosa et al., 2020] for a recent variant) to obtain a local optimum in a more restricted search space. Moreover, a complete second-order model which incorporates all connected and non-connected structures can be trained with the developed method, with which we can study the usefulness of different second-order strcutures over the performances.

CHAPTER 6
GENERAL NON-LINEAR MODEL FOR DEPENDENCY PARSING

6.1 Introduction

In Chapter 4, a mixture-of-experts (MOE) is used to approximate a non-linear function with a mixture of linear experts. The method shows that by applying a simple averaging, the performance of parsing augments by decreasing the variance of the system. In Chapter 5, we construct a higher-order polynomial score function for dependency parsing. The score function is basically a sum of score of all possible structures (from first to higher-order), and approximate solutions can be efficiently calculated thanks to the specific form of the function.

Both methods use specific hypothesis to make the problem solvable. In the first method, MOE guarantees simple calculation of the loss function, where the negative log-likelihood can be calculated by taking the log value over the weighted sum of experts' probabilities. The second method uses polynomial score function, which calculates explicitly the score of all possible structures. Thus, approximate methods like MFVI or coordinate ascent can be used. The hypothesis makes the problem solvable, but also add restrictions over the use of modern NLP techniques. As far as we know, all graph-based Dependency Parsing models calculate explicitly the score of structures over the trees, which use either MLPs or affines functions (biaffine, triaffine, etc)[Kiperwasser and Goldberg, 2016, Dozat and Manning, 2017, Zhang et al., 2020a, Wang and Tu, 2020].

In this chapter, we consider the case where we add no additional restrictions over the score function. Precisely, we only assume that the score function $S(x, y; \Theta)$ is a non-linear differentiable function to the variable y.

We mention that graph-based dependency parsing can be viewed as EBMs (see Chapter 2, 3 for details). Using general non-linear energy function constructed with neural networks is not new in EBMs. As is mentioned in Section 2.2 Chapter 2, [Belanger and McCallum, 2016] have proposed a concrete framework of EBM call SPEN. SPEN calculates the energy function as a sum of the local energy and the global energy. The local energy is limited to be linear function while the global energy can be general non-linear functions.

Minimization of the energy function is unavoidable for learning and inference of EBMs with hinge loss. [Belanger et al., 2017] propose to use an end-to-end method on SPEN by unrolling gradient-based optimization [Domke, 2012]. The method shows good performance on image denoising and Semantic Role Labeling (SRL). They also tested the Input Convex

Neural Network (ICNN) [Amos et al., 2017] to make the energy function convex. Although convex energy functions guarantee the convergence to the global minimum, parameter and architecture constraints of ICNN limit the capacity of the neural network and gives worse results. To avoid applying several loops in gradient-based methods, [Tu and Gimpel, 2018, Tu et al., 2020b] propose to train an inference network with the goal to minimize the energy function. The output of the inference is the approximate prediction which aims to minimize the energy. The method is shown to have good performances on multi-label classification.

Besides end-to-end learning and inference network which use basically neural networks for optimization, optimization methods like FW algorithm (FW) [Frank and Wolfe, 1956] can be used to directly optimize the energy. FW is suitable for sparse optimization problems [Berrada et al., 2018, Ma and Li, 2020, Tsai and El Ghaoui, 2020], where we find that the matrix representation of parse tree is exactly highly sparse (for every column of the matrix, there is one position equals 1, while others equal 0). In section 6.3.1, we present our work to adapt FW for dependency parsing.

Instead of training EBMs with hinge loss, EBMs can be transformed as a probabilistic model with the hypothesis: $p(\dots) \propto \exp(-E(\dots))$, where $\dots$ represent the variables of the energy function. With discussion in Section 2.3 Chapter 2, calculating directly the loss function with negative log-likelihood could be intractable, but we can estimate the gradient of the loss function by sampling from the distribution $p(\dots)$. Sampling with non-linear energy functions is generally intractable. To solve the problem, [Hinton, 2002] propose contrastive divergence, which use MCMC methods to approximate sampling from the model distribution $p(\dots)$. [Du et al.] shows that by estimating the ignored KL-divergence in contrastive divergence, the variance of sampling can be highly reduced. Besides MCMC, [Du and Mordatch, 2019] propose to use langevin dynamics [Welling and Teh, 2011] for sampling. We mention that langevin dynamics is more adapted for continue data (like images), but does not suit well discrete data (like structure predictions).

The chapter is organized as follows: In section 6.2, we present the learning and inference with the general non-linear model. We present explicitly the problems needed to be solved for applying learning and inference. In section 6.3.1, we present the FW algorithm [Frank and Wolfe, 1956] and its adaptations for dependency parsing with general non-linear model. We mention that learning with FW is mostly adapted for learning hinge loss. However, [Krishnan et al., 2015] present the use of FW for marginal inference, which may inspire future work of FW for probabilistic model of dependency parsing. In section 6.4, we present the probabilistic inference network, with which we can train a probabilistic model with assumption $p(y|x;\Theta) \propto \exp(S(x,y;\Theta))$. We present finally the conclusions in section 6.5.

6.2 Learning and Inference with General Non-linear Model

We assume that the score function for the sentence tree pair (x, y) is $S(x, y; \Theta)$, with Θ vector of parameters of the model and $S(x, y)$ a differentiable function to the variable y.

Similar as Equation (2.2), inference aims to find the parse which maximizes the score:

$$y = \underset{y' \in \mathcal{Y}_x}{\operatorname{argmax}} S(x, y'; \Theta)$$

Following discussions in Section 2.3 Chapter 2, we can use hinge loss or negative log-likelihood to train the model. For simplicity, we consider only one (sentence, parse) pair (x, y) while a complete calculation of loss is an averaging of $(x, y) \in (X, Y)$, with (X, Y) a subset (batch) of training data.

Hinge loss Hinge loss can be written as:

$$L(\Theta) = [\max_{y' \in \mathcal{Y}_x} (S(x, y'; \Theta) - S(x, y; \Theta) + \Delta(y, y'))]_+$$

with (x, y) the reference sentence and its corresponding parse. The crucial problem is to solve the maximization:

$$\max_{y' \in \mathcal{Y}_x} S(x, y'; \Theta) + \Delta(y, y')$$

with

$$\Delta(y, y') = \sum_{h,d \in [n]} (1 - y_{h,d}) y'_{h,d} + (1 - y'_{h,d}) y_{h,d}$$

the linear form of hamming distance between reference parse y and predicted parse y'.

Thus the difficulty for solving the maximization of hinge loss is the same as the inference (maximizing the score to the parse tree y). This is because if $S(x, y'; \Theta)$ is a linear function to the variable y', then $S(x, y'; \Theta) + \Delta(y, y')$ is still linear. If $S(x, y'; \Theta)$ is convex or generally non-linear to the variable y', $S(x, y'; \Theta) + \Delta(y, y')$ is still convex or generally non-linear. Thus methods work for the inference also work for the maximization of hinge loss. In the following discussions, we refer to inference when talking about optimization with hinge loss.

Negative Log-Likelihood Following equation (2.6), negative log-likelihood is:

$$\begin{aligned} L(\Theta) &= -\log p(y|x;\Theta) \\ &= -\log \frac{\exp(S(x,y;\Theta))}{\sum_{y'\in\mathcal{Y}_x}\exp(S(x,y';\Theta))} \\ &= -S(x,y;\Theta) + \log \sum_{y'\in\mathcal{Y}_x}\exp(S(x,y';\Theta)) \\ &= -S(x,y;\Theta) + \log Z(x,\Theta) \end{aligned}$$

where $\log Z(x;\Theta) = \log \sum_{y'\in\mathcal{Y}_x}\exp(S(x,y';\Theta))$. The crucial problem is to calculate the log partition function, which is intractable for general non-linear score functions.

In conclusion, learning and inference with general non-linear model require to solve either a maximization problem, or calculate (estimate) the partition function.

6.3 Frank-Wolf Algorithm

In this section, we present the FW algorithms and its variants for dependency parsing. The goal is to use FW for solving the inference problem directly. We present the basis of FW algorithm, the adaptation of FW for dependency parsing and proper improvements to ensure better convergence.

6.3.1 Background of Frank-Wolfe Algorithm

Frank-Wolfe algorithm (FW) can be used to solve the following convex optimization problem:

$$\min_{x\in\mathcal{M}} f(x) \tag{6.1}$$

where $\mathcal{M}$ is a compact convex set in a vector space, $f : \mathcal{M} \to \mathbb{R}$ is a convex, differentiable real-valued function [Boyd et al., 2004].

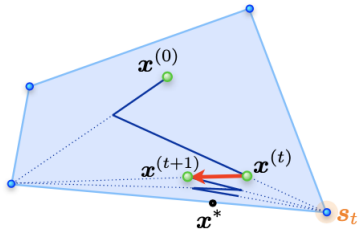

Figure 6.1: Basic Frank-Wolfe Algorithm [Lacoste-Julien and Jaggi, 2015]

To solve the problem, FW applies a linear approximation of the function and moves towards a minimizer of the linear function in the same domain $\mathcal{M}$. The basic FW algorithm is described in Alg 7 In each iteration of FW, we take the first order Taylor expansion of the

Algorithm 7: Basic Frank-Wolfe Algorithm

1 **Initialization**: $x^{(0)} \in \mathcal{M}$, $\epsilon > 0$;
2 **for** $t \leftarrow 0$ **to** T **do**
3 Let $s_t := \text{argmin}_{s \in \mathcal{M}} \langle s, \nabla f(x) \rangle$ and $d_t^{\text{F}} := s_t - x^{(t)}$; `// linear sub-problem`
`/* terminal condition */`
4 **if** $g_t^F := \langle d_t^F, -\nabla f(x^{(t)}) \rangle \leq \epsilon$ **then** terminal
5 **Return** $x^{(t)}, f(x^{(t)})$;
6 **end**
7 Let $d_t := d_t^{\text{F}}$, $\gamma_{\max} = 1$;
8 Line Search: $\gamma_t = \text{argmin}_{\gamma \in [0, \gamma_{\max}]} f(x^{(t)} + \gamma d_t)$; `// step-size determination`
9 Update $x^{(t+1)} := x^{(t)} + \gamma_t d_t$; `// update of current solution`
10 **end**
11 **Return** $x^{(T+1)}, f(x^{(T+1)})$;

function $f(x)$ and seek to minimize the the linear function in domain $\mathcal{M}$. The solution is denoted as s_t, with which we can calculate a descending direction $d_t^{\text{F}} = s_t - x_t$ called FW direction (line 3 in Alg 7). d_t can be used to decrease the function with proper step-size and it is easy to be calculated even with complex domain because only a linear function needs to be minimized. The FW direction can be used to not only minimize the function value, but also estimate the upper bound of gap between the current function value and the optimum, with $g_t^{\text{F}} = \langle d_t^{\text{F}}, -\nabla f(x^{(t)}) \rangle$. It is an upper bound of $f(x^{(t)}) - f(x^*)$, with x^* is the optimal solution because

$$g_t^{\text{F}} := \langle d_t^{\text{F}}, -\nabla f(x^{(t)}) \rangle \geq \langle x^* - x^{(t)}, -\nabla f(x^{(t)}) \rangle \geq f(x^{(t)}) - f(x^*)$$

The first inequality comes from the definition of d_t^{F} (line 3 in Alg 7) and the second inequality comes from the property of the convex function [Boyd et al., 2004]. Thus, by limiting the upper bound of the gap, we can well limit the gap between $f(x^{(t)})$ and $f(x^*)$.

For basic FW algorithm, the descending direction is set as d_t^{F} (see line 7 of Alg 7. Variants of FW use different descending direction, but d_t^{F} is still used to estimate the upper bound of gap). The step-size is determined with line search in line 8 and the update of new solution is in line 9.

For basic FW algorithm, update in line 9 shows that $x^{(t+1)}$ is a convex combination of

$x^{(t)}$ and s_t:

$$x^{(t+1)} = x^{(t)} + \gamma_t d_t = (1 - \gamma_t)x^{(t)} + \gamma_t s_t$$

Recursively, any $x^{(t)}$ can be written as a convex combination of at most $T + 2$ points $S := \{x^{(0)}, s_0, s_1, ..., s_T\} \subseteq \mathcal{M}$:

$$x^{(t)} = \alpha_0 x^{(0)} + \sum_{i=1}^{t} \alpha_i s_{i-1}$$

with $\alpha_i \geq 0, \forall i \in [t]$ and $\sum_{i=0}^{t} \alpha_i = 1$.

We call points in the set S atoms of $x^{(t)}$ and we call atoms with $\alpha_i > 0$ the active atoms. In the following discussions on variants of FW algorithm, the set of active atoms is used to calculate better descending directions.

FW algorithm can guarantee the convergence to global optimum for convex problems. For non-convex problems, we can still obtain a local optimum with FW [Frank and Wolfe, 1956, Boyd et al., 2004].

Variants of Frank-Wolfe Algorithm

As is shown in Figure 6.1, frank-wolfe zig-zags if the optimal point lies in the boundary [Lacoste-Julien and Jaggi, 2015]. This is because the current solution of basic FW algorithm jumps between two points when approaching the optimum, which makes the convergence slow. Two Variants of Frank-Wolfe algorithm can be used to solve the zig-zag problem.

Away-step Frank-Wolfe Algorithm [Wolfe, 1970] $x^{(t)}$ in each iteration of FW can be viewed as a convex combination of active atoms. One method to deal with the zig-zag problem is to make the solution move away from *bad* active atom. This is realized by defining an away direction: Suppose that for iteration t, the set of active atoms for $x^{(t)}$ is S. The away direction is defined as:

$$d_t^{\mathrm{A}} = x^t - v_t, \text{ with } v_t = \underset{v \in S}{\operatorname{argmax}} \langle v, \nabla f(x) \rangle$$

v_t is the active atom which gives the highest value over the first order Taylor expansion of $f(x)$, and thus can be viewed as a *bad* atom. When moving away from the *bad* atom gives more diminution over the linear approximation than the FW direction, i.e. $\langle d_t^{\mathrm{A}}, -\nabla f(x) \rangle > \langle d_t^{\mathrm{F}}, -\nabla f(x) \rangle$, the away direction d_t^{A} is used instead of the FW direction d_t^{F}. The detailed away-step frank-wolfe algorithm is shown in Alg 8.

From line 7 to line 12, the descending direction is determined by comparing the potential diminution brought from the the FW direction and the away-step direction. In order to calculate the away-step direction d_t^{A}, we need to know the set of active atoms (S) of current

Algorithm 8: away-step Frank-Wolfe Algorithm

Initialization: $x^{(0)} \in \mathcal{M}$, $S = \{x^{(0)}\}$, $\alpha = \{\alpha_v = 1 | v \in S\}$, $\epsilon > 0$;
for $t \leftarrow 0$ **to** T **do**
  Let $s_t := \text{argmin}_{s \in \mathcal{M}} \langle s, \nabla f(x) \rangle$ and $d_t^{\text{F}} := s_t - x^{(t)}$; `// linear sub-proble,`
  `/* terminal condition */`
  if $g_t^F := \langle d_t^F, -\nabla f(x^{(t)}) \rangle \leq \epsilon$ **then**
    Return$x^{(t)}, f(x^{(t)})$;
  end
  `/* descending direction decision */`
  Let $v_t = \text{argmax}_{v \in S} \langle v, \nabla f(x) \rangle$ and $d_t^{\text{A}} = x^{(t)} - v_t$;
  if $\langle d_t^A, -\nabla f(x) \rangle > \langle d_t^F, -\nabla f(x) \rangle$ **then**
    $d_t := d_t^{\text{A}}$, and $\gamma_{\max} := \alpha_{v_t} / (1 - \alpha_{v_t})$;
  else
    $d_t := d_t^{\text{F}}$, and $\gamma_{\max} := 1$;
  end
  Line Search: $\gamma_t = \text{argmin}_{\gamma \in [0, \gamma_{\max}]} f(x^{(t)} + \gamma d_t)$; `// step-size determination`
  `/* update of active atoms and weights */`
  if d_t ***is*** d_t^A **then**
    Update $\alpha_{v_t} := (1 + \gamma_t)\alpha_{v_t} - \gamma_t$ and $\alpha_v := (1 + \gamma_t)\alpha_v, \forall v \in S^{(t)} \backslash \{v_t\}$;
  else
    if $s_t \in S$ **then**
      Update $\alpha_{s_t} := (1 - \gamma_t)\alpha_{s_t} + \gamma_t$ and $\alpha_v := (1 - \gamma_t)\alpha_v, \forall v \in S \backslash \{s_t\}$;
    else
      Update $S := S \cup \{s_t\}$;
      Update $\alpha := \alpha \cup \{\alpha_{s_t} := \gamma_t\}$ and $\alpha_v := (1 - \gamma_t)\alpha_v, \forall v \in S \backslash \{s_t\}$;
    end
  end
  Pop $v \in S, \alpha_v \in \alpha$ with $\alpha_v = 0$;
  Update $x^{(t+1)} := x^{(t)} + \gamma_t d_t$; `// update of current solution`
end
Return $x^{(T+1)}, f(x^{(T+1)})$;

state $x^{(t)}$ and their corresponding weights (α). The update of the active atoms and the corresponding weights are shown in line 14 to line 24.

Visually (Figure 6.2), away-step FW helps the convergence by moving away from *bad* atoms.

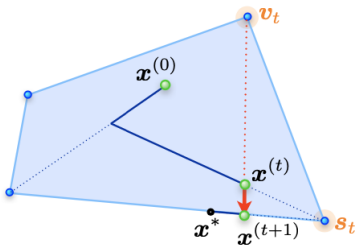

Figure 6.2: away-step Frank-Wolfe Algorithm [Lacoste-Julien and Jaggi, 2015]

Pairwise Frank-Wolfe Algorithm Instead of making a comparison of away-step direction and the FW direction, pairwise FW algorithms moves away from the *bad* atom while moving to the minimizer of the linear approximation s_t. A pairwise direction is calculated as: $d_t^P := s_t - v_t$ with $v_t := \text{argmax}_{v \in S} \langle v, \nabla f(x) \rangle$ the *bad* atom. In fact, it is a combination of the FW direction and the away-step direction: $d_t^{\text{P}} = d_t^{\text{F}} + d_t^{\text{A}}$. The detailed pairwise FW algorithm is shown in Alg 9

The update direction direction is calculated in line 7 with the pairwise direction d_t^{P}. As the *bad* atom need to be determined in each iteration, we also need to update the set of active atoms and their corresponding weights, which is shown in line 9 to line 15.

Visually (Figure 6.3), pairwise FW algorithm also accelerate the convergence by moving to s_t the minimizer s_t and moving away from bad atom v_t. In practice, pairwise FW is simpler to implement than away-step FW and it has in general better performances than away-step FW [Lacoste-Julien and Jaggi, 2015, Pedregosa et al., 2020]. In the following discussions, we focus on either basic FW or pairwise FW algorithms. It is also worthwhile to

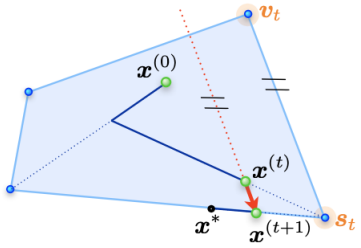

Figure 6.3: away-step Frank-Wolfe Algorithm [Lacoste-Julien and Jaggi, 2015]

note that for away-step and pairwise FW, the maximum step-size is limited by the weight of

Algorithm 9: Pairwise Frank-Wolfe Algorithm

Initialization: $x^{(0)} \in \mathcal{M}$, $S = \{x^{(0)}\}$, $\alpha = \{\alpha_v = 1 | v \in S\}$, $\epsilon > 0$;
for $t \leftarrow 0$ **to** T **do**
  Let $s_t := \text{argmin}_{s \in \mathcal{M}} \langle s, \nabla f(x) \rangle$ and $d_t^{\text{F}} := s_t - x^{(t)}$; `// linear sub-problem`
  `/* terminal condition */`
  if $g_t^F := \langle d_t^F, -\nabla f(x^{(t)}) \rangle \leq \epsilon$ **then**
    Return$x^{(t)}, f(x^{(t)})$;
  end
  `/* pairwise direction */`
  Let $v_t = \text{argmax}_{v \in S} \langle v, \nabla f(x) \rangle$, $d_t := d_t^{\text{P}} = s_t - v_t$ and $\gamma_{\max} := \alpha_{v_t}$;
  Line Search: $\gamma_t = \text{argmin}_{\gamma \in [0, \gamma_{\max}]} f(x^{(t)} + \gamma d_t)$; `// step-size determination`
  `/* update of active atoms and weights */`
  if $s_t \in S$ **then**
    Update $\alpha_{s_t} := \alpha_{s_t} + \gamma_t$ and $\alpha_{v_t} := \alpha_{v_t} - \gamma_t$;
  else
    Update $S := S \cup \{s_t\}$;
    Update $\alpha := \alpha \cup \{\alpha_{s_t} := \gamma_t\}$ and $\alpha_{v_t} := \alpha_{v_t} - \gamma_t\}$;
  end
  Pop $v \in S, \alpha_v \in \alpha$ with $\alpha_v = 0$;
  Update $x^{(t+1)} := x^{(t)} + \gamma_t d_t$; `// update of current solution`
end
Return $x^{(T+1)}, f(x^{(T+1)})$;

the *bad* active atom. This is because the greatest extend to move away from the *bad* atom is to make its weight to zero. Larger step-size may make the solution out of the domain $\mathcal{M}$.

In the next section, we discuss the adaptation of FW algorithms for dependency parsing. We show the adaptation with mainly the basic FW algorithm and the adaptation can be applied in the same way over the variants of FW.

6.3.2 Adaptation of Frank-Wolfe Algorithm for Dependency Parsing

Following previous notations, we simplify $S(x, y; \Theta)$ to $S(y)$ because both the inference and maximization problem in hinge loss concern only the variable y.

Adaptation of the Optimization Problem As the linear function $\Delta(y, y')$ does not change the difficulty of optimization, it is sufficient to consider the following problem:

$$\min_{y \in \mathcal{Y}} -S(y)$$

where $\mathcal{Y}$ is the set of all possible parses.

$\mathcal{Y}$ is a finite discrete set while in equation (6.1), we require the domain $\mathcal{M}$ to be a compact convex set. Thus, the first adaptation is to expand the discrete set $\mathcal{Y}$ to a compact convex set. This can be realized by expanding $\mathcal{Y}$ to its convex hull with convex combination [Boyd et al., 2004]:

$$\mathcal{M} := \text{conv}(\mathcal{Y}) = \{ \sum_{y' \in \mathcal{Y}} \omega_{y'} y' | \omega_y \geq 0, \forall y \in \mathcal{Y}; \sum_{y \in \mathcal{Y}} \omega_y = 1 \}$$

Remark that by using $\text{conv}(\mathcal{Y})$, the solution of FW may probably be a dense matrix which does not belong to the original discrete set $\mathcal{Y}$. We present later the projection of the dense matrix to its *nearest* discrete point in $\mathcal{Y}$.

Adaptation of Frank-Wolfe Direction For each iteration of FW algorithm, a linear sub-problem needs to be solved (see Alg 7 line 3):

$$s_t := \operatorname*{argmin}_{s \in \mathcal{M}} \langle s, \nabla f(x) \rangle$$

For dependency parsing, this can be adapted as:

$$s_t := \operatorname*{argmax}_{s \in \text{conv}(\mathcal{Y})} \langle s, \nabla S(y^{(t)}) \rangle$$

Observe that maximizing a linear function over a simplex always gives solution of a vertex [Boyd et al., 2004]. Thus, solving the linear maximization problem in $\text{conv}(\mathcal{Y})$ is equivalent

to solving the problem directly in $\mathcal{Y}$:

$$s_t := \underset{s \in \mathcal{Y}}{\operatorname{argmax}} \langle s, \nabla S(y^{(t)}) \rangle$$

The problem changes to finding the tree which maximizes a linear score function $\langle s, \nabla S(y^{(t)}) \rangle$. Thus, decoding algorithms like Eisner or Chu-Liu-Edmonds can be used for solving the problem. For clarity, we note it as:

$$s_t := \text{DA}(\nabla S(y^{(t)}))$$

where DA represents Decoding Algorithms of Eisner or Chu-Liu-Edmonds, with $\nabla S(y^{(t)})$ the score of arcs.

6.3.3 Step-size determination with Modified Backtracking Line-Search

FW algorithm and its variants require line search to determine the step-size:

$$\gamma_t = \underset{\gamma \in [0, \gamma_{\max}]}{\operatorname{argmin}} f(x^{(t)} + \gamma d_t)$$

Linear search is inefficient because this requires to evaluate several times the function values with different step-sizes to make the decision.

To make line search more efficient, [Pedregosa et al., 2020] propose the backtracking line-search to determine efficiently a proper step-size. The method is based on the assumption: the function to minimize $f(x)$ is L-smooth, i.e. it is differentiable and its gradient is L-Lipschitz continuous

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y||, \forall x, y \in \mathcal{M}$$

where $L > 0$ is the Lipschitz estimate parameter.

To determine a proper step-size γ for the function $f(x^{(t)} + \gamma d_t)$, they consider its quadratic Taylor approximation to the variable γ, and with the previous assumption, the quadratic function can be written as:

$$Q_t(\gamma, L) = f(x^{(t)}) - \gamma g_t + \frac{\gamma^2 L}{2} ||d_t||^2$$

where $g_t = \langle d_t, -\nabla f(x) \rangle$.

Thus, an optimum step-size can be calculated by minimizing the quadratic approxima-

tion:

$$\gamma_t = \underset{\gamma}{\operatorname{argmin}}\, Q_t(\gamma, L)$$

As the step-size is limited by $\gamma_{\max}$, we have

$$\gamma_t = \min(\frac{g_t}{L||d_t||^2}, \gamma_{\max}) \tag{6.2}$$

Backtracking line-search requires that a sufficient decrease condition should be satisfied:

$$f(x^{(t)} + \gamma_t d_t) \leq Q_t(\gamma_t, L) \tag{6.3}$$

which can guarantee $f(x^{(t)} + \gamma_t d_t) \leq f(x^{(t)})$ [Pedregosa et al., 2020], i.e. the step-size satisfying the condition gives the same function value in the worst case.

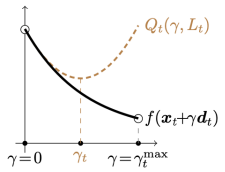

Figure 6.4: Sufficient Decrease Condition [Pedregosa et al., 2020]

Remark that the algorithm can always satisfy the condition by increasing the Lipschitz estimated parameter L[Pedregosa et al., 2020]. Note that the step-size is inversely proportional to L (see Equation 6.3). To have larger step-size, the smallest L satisfying the condition is used, which can be approximated by iteratively increasing L.

The backtracking line-search is present in details in Alg 10, with basic FW with backtracking presented in details in Alg 11. Although there is a loop in backtracking to determine the Lipschitz estimated parameter L, [Pedregosa et al., 2020] shows that a fairly small number of iterations is sufficient to satisfy the sufficient decrease condition.

The Lipschitz estimate parameter L increases until the sufficient decreasing condition is satisfied. This may lead to a small step-size which makes the algorithm too slow and even stop ($\gamma \approx 0$) later. In fact, there is no need to satisfy the sufficient decrease condition to guarantee the diminution of the function value. In fact, we only need to guarantee $f(x^{(t)} + \gamma_t d)_t) \leq f(x^{(t)})$. Inspired from the original line search which takes the step-size that minimizes the function value, all we need to do is that during the loop of backtracking, we choose the Lipschitz estimate parameter L which gives the smallest function value. In this case, we can well guarantee that $f(x^{(t)} + \gamma_t d)_t) \leq f(x^{(t)})$. Moreover it is possible to have a smaller Lipschitz estimate parameter L and a larger step-size γ, which can help to accelerate the convergence. The modified backtracking algorithm is presented in Alg 12.

Algorithm 10: Backtracking with Line-Search

Input: $f, d_t, x^{(t)}, g_t, L, \gamma_{\max}$;
Initialization: $\tau > 1, \eta \leq 1$;
Let $L := \eta L$;
Let $\gamma := \min(\frac{g_t}{L||d_t||^2}, \gamma_{\max})$; `// update step-size`
`/* sufficient decreasing condition */`
while $f(x^{(t)} + \gamma d_t) > Q_t(\gamma, L)$ **do**
    Let $L := \tau L$; `// increase` L
    Let $\gamma := \min(\frac{g_t}{L||d_t||^2}, \gamma_{\max})$; `// update step-size`
end
Return γ, L;

Algorithm 11: Basic Frank-Wolfe Algorithm with Backtracking Line-Search

Initialization: $x^{(0)} \in \mathcal{M}$, $\epsilon > 0$, $L := L_0 > 0$;
for $t \leftarrow 0$ **to** T **do**
    Let $s_t := \text{argmin}_{s \in \mathcal{M}} \langle s, \nabla f(x) \rangle$ and $d_t^{\text{F}} := s_t - x^{(t)}$; `// linear sub-problem`
    `/* terminal condition */`
    if $g_t^F := \langle d_t^F, -\nabla f(x^{(t)}) \rangle \leq \epsilon$ **then**
        Return $x^{(t)}, f(x^{(t)})$;
    end
    Let $d_t := d_t^{\text{F}}$, $g_t := g_t^{\text{F}}$, $\gamma_{\max} = 1$;
    `/* step-size determination with backtracking line-search */`
    Backtracking Line-Search: $\gamma_t, L = \text{Backtracking}(f, d_t, x^{(t)}, g_t, L, \gamma_{\max})$;
    Update $x^{(t+1)} := x^{(t)} + \gamma_t d_t$; `// update of current solution`
end
Return $x^{(T+1)}, f(x^{(T+1)})$;

Algorithm 12: Modified Backtracking with Line-Search

Input: $f, d_t, x^{(t)}, g_t, L, \gamma_{\max}$;
Initialization: $\tau > 1$, $\eta \leq 1$, $L^* := +\infty$, $\gamma^* := 0$, $v^* = f(x^{(t)})$;
Let $L := \eta L$;
Let $\gamma := \min(\frac{g_t}{L||d_t||^2}, \gamma_{\max})$; `// update step-size`
`/* sufficient decreasing condition */`
while $f(x^{(t)} + \gamma d_t) > Q_t(\gamma, L)$ **do**
    `/* selection of step-size with smallest function value */`
    if $f(x^{(t)} + \gamma d_t) < v^*$ **then**
        Let $L^* := L$, $\gamma^* := \gamma$, $v^* = f(x^{(t)} + \gamma d_t)$;
    end
    Let $L := \tau L$;
    Let $\gamma := \min(\frac{g_t}{L||d_t||^2}, \gamma_{\max})$; `// update step-size`
end
`/* selection of step-size with smallest function value */`
if $f(x^{(t)} + \gamma d_t) < v^*$ **then**
    Let $L^* := L$, $\gamma^* := \gamma$, $v^* = f(x^{(t)} + \gamma d_t)$;
end
Return γ^*, L^*;

Restart Restart has already been used in optimization algorithm to accelerate the convergence [Fukunaga, 1998, Jansen, 2002] of algorithms. Particularly for non-convex functions, restart may help the current solution to jump out of the local minimum. This is particularly useful for dependency parsing with general non-linear score functions because the score function is probably non-convex. One restart strategy is that when the update leads to an increase of the objective function value or no change, we restart the algorithm with the current state $x^{(t)}$ as the initial state. This type of restart is called adaptive restart [Kim and Fessler, 2018]. Restart can also be done every $k > 1$ iterations and is called fixed restart [Kim and Fessler, 2018]. We propose to use adaptive restart for FW algorithm with modified backtracking. As the modified backtracking can guarantee the diminution of the function value, we set the restart condition as: the step-size γ_t becomes too small, i.e. $\gamma_t \leq \delta$, with $\delta > 0$, the acceptable minimum step-size.[1].

For restart with basic FW algorithm, the initial point needs to be reset as the current state $x^{(t)}$. However, this is not enough because backtracking may still give a small step-size. Equation (6.2) shows that the step-size is inversely proportional to L. If the small

1. The restart condition can also be $f(x^{(t)}) - f(x^{(t+1)})$ becomes too small, which is equivalent to restarting when the step-size becomes too small

step-size is caused by a large value of L, restart may not be useful. To avoid this problem, we reset $L := L_0$ with L_0 the initial Lipschitz estimate parameter, which is usually a small value. In this case, restart may accelerate the convergence of the basic FW algorithm with backtracking by restarting with a large step-size.

For variants of FW algorithms, besides the reinitialization with $x^{(t)}$ and the reset $L := L_0$, we also need to reinitialize the set of active atoms to $S := \{x^{(t)}\}$ and their weights to $\alpha := \{\alpha_v = 1 \| v \in S\}$. This is because the step-size for variants of FW is also controlled by the weight of *bad* atom. When the weight of *bad* atom becomes small, the upper bound of the step-size $\gamma_{\max}$ will also become small for variants of FW, which may limit the usefulness of restart if not reinitialized. Precisely, for away-step FW with away step directed used for update, we have (see Alg 8 line 9):

$$\gamma_{max} = \frac{\alpha_{v_t}}{1 - \alpha_{v_t}}$$

For pairwise FW, we have (see Alg 9 line 7):

$$\gamma_{max} = \alpha_{v_t}$$

where v_t the *bad* atom and α_{v_t} its corresponding weight. When $\alpha_{v_t} \to 0$, $\gamma_{\max \to 0}$ for both variants of FW. Thus besides the reinitialization of the initial state and the Lipschitz estimated parameter, we also need to reinitialize the active atoms set S and their weights α for variants of FW to guarantee possible larger step-size brought by restart.

Early Stopping In machine learning, early stopping is a strategy to stop in advance the training when it brings no improvement (or worse performances) over the development data. Similarly, the restart strategy used in FW introduces naturally a early stopping strategy. This is because when restart brings no change (the step-size γ_t is still too small), there is no need to do more restart and we can terminate directly the algorithm.

We present the restart and early stopping for the basic frank-wolfe algorithm with backtracking line-search in Alg 13. The method can also be applied over the variants of FW algorithm by adding reinitialization over the active atom set S and its corresponding weights α.

6.3.4 Projection of Dense Solution

As is mentioned in the beginning of the section, FW for dependency parsing gives solutions in the space $\text{conv}(\mathcal{Y})$, which is in fact a dense matrix. We propose to project y to its 'nearest'

Algorithm 13: Basic Frank-Wolfe Algorithm with Backtracking Line-Search, Restart and Early Stopping

Initialization: $x^{(0)} \in \mathcal{M}$, $\epsilon, \delta > 0$, $L := L_0 > 0$, RESTART = False;
for $t \leftarrow 0$ **to** T **do**
  Let $s_t := \text{argmin}_{s \in \mathcal{M}} \langle s, \nabla f(x) \rangle$ and $d_t^{\text{F}} := s_t - x^{(t)}$; `// linear sub-problem`
  `/* terminal condition */`
  if $g_t^F := \langle d_t^F, -\nabla f(x^{(t)}) \rangle \leq \epsilon$ **then**
    Return $x^{(t)}, f(x^{(t)})$;
  end
  Let $d_t := d_t^{\text{F}}$, $g_t := g_t^{\text{F}}$, $\gamma_{\max} = 1$;
  `/* step-size determination with backtracking line-search */`
  Backtracking Line-Search: $\gamma_t, L = \text{Backtracking}(f, d_t, x^{(t)}, g_t, L, \gamma_{\max})$;
  `/* restart condition */`
  if $\gamma_t \leq \delta$ **then**
    `/* early stopping */`
    if *RESTART* **then**
      Return $x^{(t)}, f(x^{(t)})$;
    else
      Let $L := L_0$, RESTART = True;
    end
  else
    Let RESTART = False;
  end
  Update $x^{(t+1)} := x^{(t)} + \gamma_t d_t$; `// update current solution`
end
Return $x^{(T+1)}, f(x^{(T+1)})$;

$y^* \in \mathcal{Y}$ by minimizing their L_1 distance:

$$y^* = \underset{y' \in \mathcal{Y}}{\operatorname{argmin}} ||y' - y||_1$$

This is equivalent to[2]:

$$y^* = \underset{y' \in \mathcal{Y}}{\operatorname{argmax}} \langle y', y \rangle$$

Thus, the projection can be well solved with Eisner or Chu-Liu-Edmonds depending on different tree structure. By using the previous notation DA representing decoding algorithms, it can be written as:

$$y^* = \mathrm{DA}(y)$$

with the dense matrix y viewed as the score of the arc.

We give the final version of basic FW algorithm for dependency parsing in Alg 14, with backtracking line-search, restart, early stopping and projection. The modified backtracking for dependency parsing is shown in Alg 15. Adaptations for variants of frank-wolfe algorithm are similar and we show a detailed adaptation for pairwise FW in Appendix D.3.

6.3.5 Conservation of the Tree Structure

In the previous Sections, we have proved that the sub-problem in each iteration of FW can be solved with decoding algorithms like Eisner or Chu-Liu-Edmonds. One theoretical concern is whether is it necessary to use Eisner or Chu-Liu-Edmonds to obtain s_t which conserves the tree structure (projective or non-projective). Or whether taking the convex hull of $\mathcal{Y}$ will break the projective or non-projective property. If the tree structure is conserved for $\mathrm{conv}(\mathcal{Y})$, then the use of decoding algorithms may be useful. If not, then it becomes unnecessary to conserve the tree structure for the linear sub-problem and we can use simpler and more efficient methods.

The following proof shows that for projective tree, it may still be useful to use Eisner for solving the sub-problem. For non-projective tree, we can well replace Chu-Liu-Edmonds with simple argmax.

We start from the definition of projective tree given in Definition 1 Chapter 2: a projective tree is an acyclic directed graph which satisfies the following two properties:

1.The in-degree of every node (except the root) equals and only equals to 1.

2.$\forall (i, j) \in y$, with $j > i + 1$, $\forall m$ with $i < m < j$, there exists a path from i to m, which only passes nodes i, j and nodes between i, j.

2. see Appendix D.1

Algorithm 14: Basic Frank-Wolfe Algorithm for Dependency Parsing, with Backtracking Line-Search, Restart, Early Stopping and Projection

Initialization: $y^{(0)} \in \mathcal{Y}$, $\epsilon, \delta > 0$, $L := L_0 > 0$, RESTART = False;
for $t \leftarrow 0$ **to** T **do**
    Let $s_t := DP(\nabla S(y^{(t)}))$ and $d_t^{\mathrm{F}} := s_t - y^{(t)}$; `// linear sub-problem`
    `/* terminal condition */`
    if $g_t^F := \langle d_t^F, \nabla S(y^{(t)}) \rangle \leq \epsilon$ **then**
        Projection: $y^* := \mathrm{DA}(y^{(t)})$;
        Return $y^*, S(y^*)$;
    end
    Let $d_t := d_t^{\mathrm{F}}$, $g_t := g_t^{\mathrm{F}}$, $\gamma_{\max} = 1$;
    `/* step-size determination with backtracking line-search */`
    Backtracking Line-Search: $\gamma_t, L = \mathrm{Backtracking}(S, d_t, y^{(t)}, g_t, L, \gamma_{\max})$;
    `/* restart condition */`
    if $\gamma_t \leq \delta$ **then**
        `/* early stopping */`
        if *RESTART* **then**
            Projection: $y^* := \mathrm{DP}(y^{(t)})$;
            Return $y^*, S(y^*)$;
        else
            Let $L := L_0$, RESTART = True;
        end
    else
        Let RESTART = False;
    end
    Update $y^{(t+1)} := y^{(t)} + \gamma_t d_t$; `// update current solution`
end
Projection: $y^* := \mathrm{DA}(y^{(T+1)})$;
Return $y^*, S(y^*)$;

Algorithm 15: Modified Backtracking for Dependency Parsing with Line-Search

Input: $S, d_t, y^{(t)}, g_t, L, \gamma_{\max}$;
Initialization: $\tau > 1$, $\eta \leq 1$, $L^* := +\infty$, $\gamma^* := 0$, $v^* = -S(y^{(t)})$;
Let $L := \eta L$;
Let $\gamma := \min(\frac{g_t}{L||d_t||^2}, \gamma_{\max})$;
while $-S(y^{(t)} + \gamma d_t) > Q_t(\gamma, L)$ **do**
    if $-S(y^{(t)} + \gamma d_t) < v^*$ **then**
        Let $L^* := L$, $\gamma^* := \gamma$, $v^* = -S(y^{(t)} + \gamma d_t)$;
    end
    Let $L := \tau L$;
    Let $\gamma := \min(\frac{g_t}{L||d_t||^2}, \gamma_{\max})$;
end
if $-S(y^{(t)} + \gamma d_t) < v^*$ **then**
    Let $L^* := L$, $\gamma^* := \gamma$, $v^* = -S(y^{(t)} + \gamma d_t)$;
end
Return γ^*, L^*;

For non-projective tree, only the first property is required.

The first property can be expressed as:

$$\sum_{i=1}^{n} y_{i,j} = 1 \quad \forall j \in \{1, ..., n\}, \forall y \in \mathcal{Y}$$

And it is simple to verify that the first property is satisfied $\forall y \in \text{conv}(\mathcal{Y})$:

$$\begin{aligned}\sum_{i=1}^{n} y_{i,j} &= \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} \omega_{y'} y'_{i,j}, \forall y \in \text{conv}(\mathcal{Y}) \\ &= \sum_{y' \in \mathcal{Y}} \omega_{y'} \sum_{i=1}^{n} y'_{i,j} \\ &= 1\end{aligned}$$

To rewrite formally the second property, we use $\mathcal{P}_i^j(m)$ to represent the set of all paths from node i to node m between i, j, which only pass nodes i, j and nodes between i, j. We note $r \in \mathcal{P}_i^j(m)$ one arc in the set and $(s, t) \in r$ one directed arc in the path from i to m. Then the second property can be represented as: if $\exists y \in \mathcal{Y}$ s.t. $y_{i,j} \neq 0$, with $j > i + 1$.

Then $\forall m$ with $i < m < j$, we have:

$$\sum_{r \in \mathcal{P}_i^j(m)} \prod_{(s,t) \in r} y_{s,t} \neq 0 \tag{6.4}$$

To prove that the second property is also satisfied $\forall y \in \text{conv}(\mathcal{Y})$, suppose $\exists y \in \text{conv}(\mathcal{Y})$ s.t. $y_{i,j} \neq 0$ with $j > i + 1$. Then as $y \in \text{conv}(\mathcal{Y})$ can be written as a convex combination of points in the set $\mathcal{Y}$ we have:

$$y_{i,j} = \sum_{y' \in \mathcal{Y}} \omega_{y'} y'_{i,j} \neq 0, \text{ with } \sum_{y' \in \mathcal{Y}} \omega_{y'} = 1, \omega_{y'} \geq 0$$

This indicates that $\exists y^* \in \mathcal{Y}$ s.t. $\omega_{y^*} > 0$ and $y^*_{i,j} \neq 0$. By reusing Equation 6.4, we have:

$$\sum_{r \in \mathcal{P}_i^j(m)} \prod_{(s,t) \in r} y^*_{s,t} \neq 0 \tag{6.5}$$

$\forall m$ with $i < m < j$.

Thus, for $y \in \text{conv}(\mathcal{Y})$ with $y_{i,j} \neq 0$, we have:

$$\begin{aligned}
\sum_{r \in \mathcal{P}_i^j(m)} \prod_{(s,t) \in r} y_{s,t} &= \sum_{r \in \mathcal{P}_i^j(m)} \prod_{(i,j) \in r} \sum_{y' \in \mathcal{Y}} \omega_{y'} y'_{s,t} \\
&\geq \sum_{r \in \mathcal{P}_i^j(m)} \prod_{(s,t) \in r} \omega_{y^*} y^*_{s,t} \\
&= \omega_{y^*} \sum_{r \in \mathcal{P}_i^j(m)} \prod_{(s,t) \in r} y^*_{s,t} \\
&\neq 0
\end{aligned}$$

The first line uses condition that $y \in \text{conv}(\mathcal{Y}$. In the second line, we replace the summation over all possible arborescence with one single arborescence y^*. As $\omega_{y'} y'_{s,t} \geq 0$, the inequality is valid. We reformulate the equation in the third line and find that it is a product of $\omega_{y^*} \geq 0$ the Equation 6.5, which does not equals 0. Thus, we prove that the second property is still conserved $\forall y \in \text{conv}(\mathcal{Y})$.

We mention that the property acyclic is not conserved for all $y \in \text{conv}(\mathcal{Y})$. This can be seen from the convex combination of two trees, with one has arc (i, j) and the other has arc (j, i).

The previous proof shows that for projective trees, all properties are satisfied $\forall y \in$

conv($\mathcal{Y}$) except for the acyclic property. This indicates that by using Eisner algorithm for the linear sub-problem in each iteration of FW, we still search in a highly structured space. However, for non-projective trees, as the second property is no longer needed, only the first property is conserved. As the acyclic property is not conserved by taking the convex hull of non-projective trees, there is no need to use Chu-Liu-Edmonds to guarantee the acyclic property for s_t. Thus when using FW for non-projective trees, we can well replace Chu-Liu-Edmonds by simply taking argmax over each column of the matrix, which is much more efficient. Note that for the projection of dense tree to its nearest discrete tree, Chu-Liu-Edmonds is still useful to guarantee the acyclic property.

In conclusion, we have:

$$\text{DA} = \begin{cases} \text{Eisner} & \text{if projective} \\ \text{argmax} & \text{if non-projective} \end{cases}$$

for solving the linear sub-problem in each iteration of FW, which can increase a lot the speed for the non-projective tree.

When it comes to projection, we still need to use:

$$\text{DA} = \begin{cases} \text{Eisner} & \text{if projective} \\ \text{Chu-Liu-Edmonds} & \text{if non-projective} \end{cases}$$

6.4 Probabilistic Inference Network

Inference network has been used for structure prediction [Tu and Gimpel, 2018, Tu et al., 2020a,b] and machine translation [Tu et al., 2020c]. In Section 2.4, we discussed learning SPEN with the inference network [Tu and Gimpel, 2018, Tu et al., 2020b] through hinge loss. The method proposes to use another network G with parameter Φ (which is called inference network) to solve the sub-problem of hinge loss, and the output of the inference network G is used as the most violating prediction. This creates in fact a saddle point optimization problem and can be expressed as:

$$\min_{\Theta} \max_{\Phi} [E(x, y; \Theta) + \Delta(y, G(x; \Phi)) - E(x, G(x; \Phi); \Theta)]_+$$

where Θ is the vector of parameters of SPEN, Φ is the vector of parameters of the inference network. $G(x; \Phi)$ is the output of inference network which is used to find the most violating prediction for hinge loss. Instead of using FW to directly infer the most violating prediction, inference network predicts it with a *fowrard pass* on G, which could be more efficient.

Using inference network with hinge loss has been well discussed in previous works [Tu and Gimpel, 2018, Tu et al., 2020b,a]. However, the use of inference network for probabilistic model is not clear. In this section, we present how to use inference network to train a non-linear probabilistic model. The main idea is that instead of using inference network to do inference (solving the minimization/maximization problem), we propose to use the inference network to learn the distribution $p(y|x;\Theta)$.

6.4.1 Learning Framework

We present the loss function used for learning the non-linear model for dependency parsing and the inference network. As the framework inference network is similar to GAN [Goodfellow et al., 2014], we use L_G (G for generator) for the loss function of the inference network and L_D (D for discriminator) for the loss function of the non-linear model for dependency parsing.

Discriminator Loss Function As discussed in Section 6.2, the main difficulty for calculating the negative log-likelihood lies in the calculation of the log partition function:

$$\log Z(\Theta) = \log \sum_{y' \in \mathcal{Y}_x} \exp(S(y';\Theta))$$

which is intractable for general non-linear score functions. With discussions in Section 2.3, instead of estimating the partition function, we can estimate directly the gradient with equation (2.7), which can be modified for dependency parsing by replacing the energy with score:

$$\frac{dL_{\text{NLL}}(\Theta)}{\partial\Theta} = \frac{\partial}{\partial\Theta}[\mathbb{E}_{y'\sim p(y|x;\Theta)} S(y';\Theta) - S(y;\Theta)]$$

Thus instead of using negative log-likelihood which may be intractable for non-linear score, we can use the following loss function which gives same gradient over the vector of parameters Θ as negative log-likelihood:

$$L_D(\Theta) = \mathbb{E}_{y'\sim p(y|x;\Theta)} S(y';\Theta) - S(y;\Theta)$$

It is still intractable to calculate the exact expectation of $S(y;\Theta)$:

$$\mathbb{E}_{y'\sim p(y|x;\Theta)} S(y';\Theta) = \sum_{y\in\mathcal{Y}} p(y|x;\Theta) S(y;\Theta)$$

Because firstly it is intractable to calculate $p(y|x;\Theta)$, and it is also intractable to calculated a weighted sum of non-linear score for all possible trees. However, the law of large num-

bers [Grimmett and Stirzaker, 2020] allows us to estimate the loss function with K trees $\{y^{(1)}, ..., y^{(K)}\}$ sampled to the distribution $p(y|x;\Theta)$, and the loss function can be estimated as:

$$L_D(\Theta) \approx \frac{1}{K}\sum_{i=1}^{K} S(y^{(i)};\Theta) - S(y;\Theta), \text{with } \{y^{(1)}, ..., y^{(K)}\} \sim p(y|x;\Theta) \tag{6.6}$$

Still as $S(y;\Theta)$ is a non-linear function to the variable y, it is difficult to sample directly to the distribution $p(y|x;\Theta)$. Inspired by Variational AutoEncoder (VAE) [Kingma and Welling, 2014], we can use another simple-to-sample distribution $q(y)$ to do the sampling while we try to approximate $q(y)$ to $p(y|x;\Theta)$. In this case, we may sill have a good estimation of the discriminator loss function with:

$$L_D(\Theta) \approx \frac{1}{K}\sum_{i=1}^{K} S(y^{(i)};\Theta) - S(y;\Theta), \text{with } \{y^{(1)}, ..., y^{(K)}\} \sim q(y)$$

Generator Loss Function Following the previous discussion, we propose to use:

$$q(y) := q(y|x;\Phi)$$

where $q(y|x;\Phi) \propto S(x, y;\Phi)$, with $S(x, y;\Phi)$ the score function of the inference network. To make $q(y|x;\Phi)$ a simple-to-sample distribution, we use arc-factored (linear) model for the inference network.

To approximate the distribution $q(y|x;\Phi)$ to the distribution $p(y|x;\Theta)$, we can use KL divergence (asymmetric) [Kullback and Leibler, 1951] or Jensen Shannon (JS) divergence (symmetric) [Manning and Schutze, 1999] as the loss function. When x is unambiguous from the context, we use $p(y;\Theta)$ and $q(y;\Phi)$ to represent separately $p(y|x;\Theta)$ and $q(y|x;\Phi)$. The calculation of the loss function is shown below:

KL Divergence q||p: As KL divergence is asymmetric, we use $\overrightarrow{L_G^{\mathrm{KL}}}(\Phi)$ to represent the loss function calculated with KL divergence with $q(y;\Phi)$ as the reference probability.

The loss function with the KL divergence can be calculated as:

$$\begin{aligned}\overrightarrow{L_G^{\mathrm{KL}}}(\Phi) &= \mathrm{D}_{\mathrm{KL}}(q(y;\Phi)||p(y;\Theta)) \\ &= \sum_{y\in\mathcal{Y}} q(y;\Phi)\log\frac{q(y;\Phi)}{p(y;\Theta)}\end{aligned}$$

This is intractable because we cannot calculate the probability $p(y;\Theta)$. Also the sum over all possible trees is intractable for general non-linear model. Inspired with equation (2.7) which estimates directly the gradient of the loss function, the gradient to the Φ can be calculated

as[3]:

$$\frac{d\overrightarrow{L_G^{\mathrm{KL}}}(\Phi)}{d\Phi} = \mathbb{E}_{y\sim q(y;\Phi)}(1+\log q(y;\Phi)-\log p(y;\Theta))\frac{\partial \log q(y;\Phi)}{\partial \Phi}$$

The only unknown term in the equation is $\log p(y;\Theta)$, which can also be estimated by using the easy to sample distribution $q(y;\Phi)$[4]:

$$\log p(y;\Theta) = S(y;\Theta) - \log \mathbb{E}_{y'\sim q(y;\Phi)}\frac{\exp(S(y';\Theta))}{q(y;\Phi)}$$

Thus, the gradient can be calculated as:

$$\frac{d\overrightarrow{L_G^{\mathrm{KL}}}(\Phi)}{d\Phi} = \mathbb{E}_{y\sim q(y;\Phi)}(1+\log q(y;\Phi)\overbrace{-S(y;\Theta)+\log \mathbb{E}_{y'\sim q(y;\Phi)}\frac{\exp(S(y';\Theta))}{q(y;\Phi)}}^{\text{Estimation of the negative log-likelihood}})\frac{\partial \log q(y;\Phi)}{\partial \Phi}$$

This indicates that we can use the following loss function as the loss function for the inference network:

$$\overrightarrow{L_G^{\mathrm{KL}}}(\vec{\Phi}) = \mathbb{E}_{y\sim q(y;\Phi)}(1+\log q(y;\Phi)\overbrace{-S(y;\Theta)+\log \mathbb{E}_{y'\sim q(y;\Phi)}\frac{\exp(S(y';\Theta))}{q(y;\Phi)}}^{\text{Estimation of the negative log-likelihood}})\log p(y;\vec{\Phi}) \tag{6.7}$$

As the gradient is only calculated over the last term, we use $\vec{\Phi}$ to represent parameters which require the calculation of the gradient, while Φ is viewed as a constant.

With K trees $\{y^{(1),\ldots,y^{(K)}}\}$ sampled from the distribution $q(y;\Phi)$, the loss of the inference can be estimated as:

$$\overrightarrow{L_G^{\mathrm{KL}}}(\vec{\Phi}) \approx \frac{1}{K}\sum_{i=1}^{K}[1+\log q(y^{(i)};\Phi) - S(y^{(i)};\Theta) + \log \frac{1}{K}\sum_{j=1}^{K}\frac{\exp(S(y^{(j)};\Theta))}{q(y^{(j)};\Phi)}]\log q(y^{(i)};\vec{\Phi})$$

KL Divergence p||q: We use $\overleftarrow{L_G^{\mathrm{KL}}}(\Theta)$ to represent another form of KL divergence which uses $p(y;\Theta)$ as the reference probability:

$$\begin{aligned}\overleftarrow{L_G^{\mathrm{KL}}}(\Phi) &= \mathrm{D}_{\mathrm{KL}}(p(y;\Theta)||q(y;\Phi)) \\ &= \sum_{y\in\mathcal{Y}} p(y;\Theta)\log\frac{p(y;\Theta)}{q(y;\Phi)}\end{aligned}$$

3. see Appendix D.2

4. see Appendix D.4

With similar deduction, we have[5]:

$$\frac{d\overleftarrow{L_G^{\mathrm{KL}}}(\Phi)}{d\Phi} = \mathbb{E}_{y\sim q(y;\Phi)} - \frac{p(y;\Theta)}{q(y;\Phi)}\frac{\partial \log q(y;\Phi)}{\partial \Phi}$$

With the previous deduction, $p(y;\Theta)$ can be calculated as:

$$p(y;\Theta) = \frac{\exp(S(y;\Theta))}{\mathbb{E}_{y'\sim q(y;\Phi)}\frac{\exp(S(y';\Theta))}{p(y';\Phi)}}$$

and another form of the loss function is:

$$\overleftarrow{L_G^{\mathrm{KL}}}(\vec{\Phi}) = \mathbb{E}_{y\sim q(y;\Phi)}\frac{-1}{q(y;\Phi)}\overbrace{\frac{exp(S(y;\Theta))}{\mathbb{E}_{y'\sim q(y;\Phi)}\frac{\exp(S(y';\Theta))}{p(y';\Phi)}}}^{\text{Estimation of the likelihood}}\log p(y;\vec{\Phi})$$

Similarily, we can estimate the loss function with K samples:

$$\begin{aligned}\overleftarrow{L_G^{\mathrm{KL}}}(\vec{\Phi}) &\approx \frac{1}{K}\sum_{i=1}^{K}\frac{-\exp(S(y^{(i)};\Theta))}{p(y^{(i)};\Phi)\frac{1}{K}\sum_{j=1}^{K}\frac{\exp(S(y^{(j)};\Theta))}{p(y^{(j)};\Phi)}}\log p(y^{(i)};\vec{\Phi}) \\ &= \sum_{i=1}^{K}\frac{-\exp(S(y^{(i)};\Theta))}{p(y^{(i)};\Phi)\sum_{j=1}^{K}\frac{\exp(S(y^{(j)};\Theta))}{p(y^{(j)};\Phi)}}\log p(y^{(i)};\vec{\Phi})\end{aligned} \tag{6.8}$$

JS Divergence The JS divergence can be calculated with:

$$L_G^{\mathrm{JS}}(\Phi) = \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(q(y;\Phi)||p(y;\Theta)) + \frac{1}{2}\mathrm{D}_{\mathrm{KL}}(p(y;\Theta)||q(y;\Phi))$$

Thus, it can be estimated by simply using the equation (6.7) and the equation (6.8):

$$L_G^{\mathrm{JS}}(\vec{\Phi}) \approx \frac{1}{2}[\overrightarrow{L_G^{\mathrm{KL}}}(\vec{\Phi}) + \overleftarrow{L_G^{\mathrm{KL}}}(\vec{\Phi})] \tag{6.9}$$

6.4.2 Finer Sampling with Metropolis-Hastings

To estimate the discriminator loss function with equation (6.6), the ideal situation is to sample directly with the distribution of discriminator $p(y;\Theta)$. As this is impossible for non-linear score function for the discriminator, we propose to sample with the simple-to-

5. see Appendix D.2

sample distribution $q(y;\Phi)$, which is the distribution of an arc-factored model. Moreover, we approximate $q(y;\Phi)$ to $p(y;\Theta)$ with KL divergence or JS divergence. However, $q(y;\Phi)$ cannot approximate exactly the distribution $p(y;\Theta)$ except that $q(y;\Phi)$ is the distribution of a linear model[6]. Thus when $q(y;\Phi)$ is non-linear, a non-zero error may exits between q and p, which can lead to inaccurate sampling.

To solve the problem, we propose to use Metropolis-Hastings algorithm (MH) [Hastings, 1970] to approximate sampling from $p(y;\Theta)$, with $q(y;\Phi)$ as the proposal distribution. MH algorithm is a MCMC method for generating a sequence of samples from a hard-to-sample distribution. The proof of validity of the algorithm is out of the scope of the thesis and can be referred in [Brooks et al., 2011]. The algorithm is described in Alg 16. With $S(y;\Phi)$ as the

Algorithm 16: Metropolis Hastings

Initialization: $y^{(0)} \sim q(y;\Phi)$, $t = 0$;
while $t < K$ **do**
  Sample $y' \sim q(y;\Phi)$; `// generate a new sample`
  Acceptance probability: $A(y', y^{(t)}) := \min(1, \frac{p(y';\Phi)}{p(y^{(t)};\Phi)} \frac{\exp(S(y^{(t)};\Theta))}{\exp(S(y');\Theta)})$;
  Sample $u \in [0, 1]$ with uniform distribution;
  if $u \leq A(y', y^{(t)})$ **then**
    Set $y^{(t+1)} := y'$; `// accept the new sample`
  else
    Set $y^{(t+1)} := y^{(t)}$; `// reject the new sample`
  end
  Set $t := t + 1$
end

score function of the inference network. To make the distribution of the inference network simple-to-sample, $S(y;\Phi)$ is designed to be linear to y. The generated sequence of samples with MH $\{y^{(1)}, ..., y^{(K)}\}$ is then used as better approximate samples to the distribution $p(y;\Theta)$.

Thus, with the loss function $L_G(\Phi)$ (estimated with either KL divergence or JS divergence), we can approximate $q(y;\Phi)$ to $p(y;\Theta)$. Although $q(y;\Phi)$ cannot approximate exactly in theory $p(y;\Theta)$, we can use MH algorithm to do the correction and thus give better samples.

6. see Appendix D.5

6.4.3 Inference

For inference with inference network, we can follow [Tu and Gimpel, 2018, Tu et al., 2020b] to use the inference network directly for the inference. As the inference network is designed as a linear model, Eisner or Chu-Liu-Edmods can be used for efficient inference.

According to the previous discussions, the inference network cannot approximate exactly the distribution of a non-linear model. Thus, we can use the solution of inference network as an initialization, and then use FW algorithm to do fine search with the non-linear model.

6.4.4 Unsolved Problem

One unsolved problem of inference network for dependency parsing is the dropout [Srivastava et al., 2014a] used in non-linear model.

To avoid overfitting, the dropout method is used in the discriminator. The distribution of the discriminator is represented as $p(y;\Theta)$. But in reality, dropout creates a different Θ_t in iteration t of training, where Θ_t is a sub-set of parameters over Θ. Thus when training the inference network, $q(y;\Phi)$ tries to approximate in fact $p(y;\Theta_t)$. Although $p(y;\Theta_t)$ is assumed to be similar as $p(y;\Theta)$, we found n practice that dropout varies the distribution a lot , which makes learning of the inference network quite unstable. The same problem happens when using inference network with hinge loss.

As far as we know, the problem of dropout is not mentioned in published articles of inference network.

6.5 Conclusions

In this section, we presented two methods (FW algorithm and probabilistic inference network) for learning and inference with general non-linear model for dependency parsing.

FW algorithm can be used for inference. For learning of the model, it is mostly adapted with hinge loss. To adapt FW for deep dependency parsing, we expanded the discrete set of trees $\mathcal{Y}$ to a compact convex set by using its convex hull $\text{conv}(\mathcal{Y})$. We showed that the sub-linear problem in each iteration of FW algorithm can be solved efficiently with Eisner or Chu-Liu-Edmonds. To avoid inefficient line search, we used backtracking line-search to determine a proper step-size with few iterations of evaluation. Moreover we proposed a modified version of backtracking by replacing the solution which satisfies the strict decrease condition with the solution which gives the highest score. Restart and early stopping strategies were added to accelerate the convergence and projection was used in the end to project the dense solution of FW in $\text{conv}(\mathcal{Y})$ back to its nearest point in $\mathcal{Y}$, with the nearest point

found by minimizing their L_1 distance. We proved that this can also be realized efficiently with Eisner or Chu-Liu-Edmonds. Finally, we showed theoretically that the convex hull $\text{conv}(\mathcal{Y})$ breaks the acyclic property of $\mathcal{Y}$ but conserves the other two properties (single head property, projective property) for projective trees. Thus the convex hull of projective trees is still highly structured and the use of Eisner for the linear sub-problem for the sub-linear problem is still necessary. For non-projective tree, only the single head property is conserved. Thus it is safe to replace Chu-Liu-Edmonds with argmax, which is much more efficient.

For probabilistic inference network, we modified the inference network method, which is originally used with hinge loss, to a probabilistic version. The original inference network can track one point (maximum or minimum of the discriminator) while the probabilistic inference network is designed to approximate the distribution of the discriminator. As the calculation of the loss function is difficult for both the discriminator (non-linear SPEN) and the generator (inference network), we proposed to calculate directly the gradient of the loss and use sampling methods to estimate the gradient. Different versions of loss function for the inference network were given with KL divergence or JS divergence. Although it is impossible to approximate exactly the distribution of a non-linear discriminator, MH algorithm can be used to give better samples.

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis, we focused on graph-based dependency parsing and studied the non-linear graph-based models under the framework of EBMs.

In Chapters 2 and 3, we introduced the background of the thesis.

- In Chapter 2, we introduced the basis of EBMs. Graph-based models can be viewed as EBMs and learning methods of EMBs can be applied on graph-based models for training. We made comparisons between FFMs and EBMs to present the advantages and difficulties of using EBMs for structure prediction. EBMs can be used to construct complex models to exploit the information of complicated structures, but may make learning and inference intractable. We presented max-margin learning and probabilistic learning for EBMs and clarify their differences and connections. We presented a concrete framework SPEN in the end of chapter 2. SPEN separate the energy of the model as local energy and global energy. The local energy is designed to explore simple relations of the structure with linear functions while the global energy is non-linear, which aims to explore complex relations in the structure. SPEN is used to construct our own non-linear models for dependency parsing, with the linear part designed to be an arc-factored model and the global part as higher-order models or general non-linear models.

- In Chapter 3, we introduced the task of dependency parsing. We introduced projective and non-projective parses, which have different constraints over the structure and require different decoding algorithms. For previous works on dependency parsing, we presented in details the arc-factored and second-order models and their corresponding inference algorithms. For more recent advances of dependency parsing with deep learning, we present the architecture of neural network used in deep graph-based models. LSTM is used for feature extraction while affine functions are used for calculate the score. Batchified inference algorithms with back-propagation was presented, which can benefit from the acceleration of modern GPUs. In the end of the chapter, we presented end-to-end learning with MFVI. The method gives approximated estimation of the probability of arcs, but can still benefit from the use higher-order structures. The Chapter is a foundations of our discussions on non-linear models.

In Chapters 3, 4 and 5, we present our contributions on non-linear graph-based models:

- We proposed a non-linear probabilistic model with Mixture-of-Experts (MoE) in Chapter 4 and we proposed efficient inference algorithms based on MBR decoding. We studied the averaging effect and the clustering of MoE for dependency parsing. The averaging effect shows that by using a simple averaging MoE, performance of parsers increases by reducing the variance of the system. For the clustering effect, we proposed to calculate a weighted sum over the experts. We proposed a stabilized method for learning weights of experts, which solves the degeneration problem in learning of MoE. In practice, we found the clustering effect achieved little but consistent augmentation of the performance over the dev data.

- In Chapter 5, we generalized the existing constrained higher-order models to a general polynomial model, which can incorporate the score of all possible structures of any order. We proposed a efficient gradient-based method for inference with coordinate ascent. As the general polynomial is non-convex, we propose to combine genetic algorithm with inference for better solutions. Moreover, we propose a linearized marginalization method for learning and inference based on the gradient. The basic idea is to approximate the polynomial model with a linear approximation around the inference solution, and then treat the linear approximation as an arc-factored model. We verify experimentally that the method shows generally better performance than learning with hinge loss, and can be used to guarantee the tree structure (projective or non-projective) with Eisner or Chu-Liu-Edmonds at inference.

- Finally we studied the general non-linear model which we do not attribute particular restrictions over the form of the score function. We proposed to introduce convex optimization method for inference and adapted FW algorithm for dependency parsing. We proved that the linear sub-problem in each iteration of FW can be solved efficiently with inference algorithms for arc-factored models. To accelerate the speed of calculation, backtracking, restart and early stopping strategies are used for faster convergence. FW is most adapted for max-margin learning. For probabilistic learning, we propose a probabilistic inference network to approximate the distribution of the general non-linear models. The calculation of the loss function is intractable for non-linear models while we proposed to estimate directly the gradient with sampling and MCMC. New loss function based on KL-divergence and JS-divergence are derived and explained in detail for training probabilistic inference network.

7.2 Future Work

Clustering with Different Experts In Chapter 4 we have studied the MoE for dependency parsing. The averaging effect is shown to be beneficial for the performance but the clustering effect does not have evident influences and gives similar weights for all experts. One possibility is that we use same type of experts for MoE, which makes the MoE hard to cluster sentences to different experts. As the stabilized training of MoE is well solved, we can well construct a MoE with different experts. For example, an MoE with 3 experts can be constructed, with the first expert assigned to be the arc-factored model, the second expert assigned to a second order model with adjacent siblings and the third expert assigned to be a second order model with grandparent. It is also possible to combine the exploration with our work in chapter 5. With the general polynomial model for parsing dependency, we can have much more types of experts than experts constructed with constrained higher-order models, which may help to build more complex MoEs. The intuition of using different types of experts in MoE is that for simple sentences, simple parsers like arc-factored model may work better while for complex sentences, parsers with higher-order structures may work better. In this case, we may be able to well separate the clusters and may have better performances than MoE with same experts.

Explore the Usefulness of Polynomial Factors In Chapter 5, we have have studied the general polynomial model, which incorporate all possible structures of any order. Although the order of the model is limited by the memory of current modern GPUs, we can study at least the general second order model, which has not only the connected structures (sibling, grandchild), but also all possible non-connected structures. With our developed learning and inference method for general polynomial models, we can study the effect of using different structures over the performances of second order models. The more concrete question can be asked as: are all second order structures equally useful, or maybe some of the second order structures are more useful than the others. If the question is well answered, it may be possible to build better but also memory saving second order models by considering some of the most useful second order structures.

New Architecture of Neural Work for Efficient Score Calculation In Chapter 6, we adapt FW algorithm for dependency parsing. FW gives local optimum for non-linear functions, and can be used for learning and inference with any non-linear score functions.

As far as we know, all SOTA results on graph-based dependency parsing use affine functions to calculate the score of the arborescence, which limits the score function to be polynomial functions. With our work on adaptation of FW for dependency parsing, it is possible

to consider general non-linear (non-polynomial) score function which uses powerful architectures of neural network other than affine functions. We should pay attention that the calculation of the score function should be efficient because FW requires several evaluations of the score function to converge. In this case, Transformer [Vaswani et al., 2017] may be a promising direction, which is powerful and may also be efficient (parallel calculation).

REFERENCES

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

Paolo Baldi. Stochastic calculus. In *Stochastic Calculus*, pages 1–14. Springer, 2017.

David Belanger and Andrew McCallum. Structured prediction energy networks. In *International Conference on Machine Learning*, pages 983–992. PMLR, 2016.

David Belanger, Bishan Yang, and Andrew McCallum. End-to-end learning for structured prediction energy networks. In *International Conference on Machine Learning*, pages 429–439. PMLR, 2017.

Leonard Berrada, Andrew Zisserman, and M Pawan Kumar. Deep frank-wolfe for neural network optimization. *arXiv preprint arXiv:1811.07591*, 2018.

D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

Andrew D. Brown and Geoffrey E. Hinton. Products of hidden markov models. In Thomas S. Richardson and Tommi S. Jaakkola, editors, *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, AISTATS 2001, Key West, Florida, USA, January 4-7, 2001*. Society for Artificial Intelligence and Statistics, 2001. URL `http://www.gatsby.ucl.ac.uk/aistats/aistats2001/files/brown143.ps`.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/K18-2005. URL `https://www.aclweb.org/anthology/K18-2005`.

Jinho D. Choi and Andrew McCallum. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://aclanthology.org/P13-1104`.

Caio Corro and Ivan Titov. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=BJlgNh0qKQ.

Michael A Covington. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, volume 1. Citeseer, 2001.

Bart Decadt, Véronique Hoste, Walter Daelemans, and Antal van den Bosch. GAMBL, genetic algorithm optimization of memory-based WSD. In *Proceedings of SENSEVAL-3, the Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 108–112, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-0827.

Yuntian Deng, Anton Bakhtin, Myle Ott, Arthur Szlam, and Marc'Aurelio Ranzato. Residual energy-based models for text generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=B1l4SgHKDH.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326. PMLR, 2012.

Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Hk95PK9le.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1128. URL https://www.aclweb.org/anthology/P18-1128.

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32, 2019.

Yilun Du, Shuang Li, Joshua Tenenbaum, and Igor Mordatch. Improved contrastive divergence training of energy-based models supplementary.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. 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)*, pages 334–343, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1033. URL https://aclanthology.org/P15-1033.

Jason Eisner. Efficient normal-form parsing for Combinatory Categorial Grammar. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 79–86, Santa Cruz, California, USA, June 1996. Association for Computational Linguistics. doi: 10.3115/981863.981874. URL https://www.aclweb.org/anthology/P96-1011.

Jason Eisner. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 54–65, Boston/Cambridge, Massachusetts, USA, September 17-20 1997. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/1997.iwpt-1.10.

Jason Eisner. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 1–17, Austin, TX, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-5901. URL https://aclanthology.org/W16-5901.

Agnieszka Falenska and Jonas Kuhn. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1012. URL https://www.aclweb.org/anthology/P19-1012.

Erick Fonseca and André F. T. Martins. Revisiting higher-order dependency parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8795–8800, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.776. URL https://www.aclweb.org/anthology/2020.acl-main.776.

Victoria Fossum and Kevin Knight. Combining constituent parsers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 253–256, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL https://aclanthology.org/N09-2064.

Charles W Fox and Stephen J Roberts. A tutorial on variational bayesian inference. *Artificial intelligence review*, 38(2):85–95, 2012.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

Alex S Fukunaga. Restart scheduling for genetic algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 357–366. Springer, 1998.

Sida Gao and Matthew R. Gormley. Training for Gibbs sampling on conditional random fields with neural scoring factors. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4999–5011, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.406. URL https://aclanthology.org/2020.emnlp-main.406.

Vaibhava Goel and William J. Byrne. Minimum bayes-risk automatic speech recognition. *Comput. Speech Lang.*, 14(2):115–135, 2000. doi: 10.1006/csla.2000.0138. URL https://doi.org/10.1006/csla.2000.0138.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris Maddison. Oops i took a gradient: Scalable sampling for discrete distributions. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3831–3841. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/grathwohl21a.html.

Nathan Green and Zdeněk Žabokrtský. Hybrid combination of constituency and dependency trees into an ensemble dependency parser. In *Proceedings of the Workshop on Innovative Hybrid Approaches to the Processing of Textual Data*, pages 19–26, Avignon, France, April 2012. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W12-0503.

Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford university press, 2020.

Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 1997.

Dick Grune and Ceriel JH Jacobs. *Parsing techniques: a practical guide*. Springer Science & Business Media, 2007.

Michael Gygli, Mohammad Norouzi, and Anelia Angelova. Deep value networks learn to evaluate and iteratively refine structured outputs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1341–1351. PMLR, 2017. URL http://proceedings.mlr.press/v70/gygli17a.html.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W09-1201`.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Thomas Jansen. On the analysis of dynamic restart strategies for evolutionary algorithms. In *International conference on parallel problem solving from nature*, pages 33–43. Springer, 2002.

Tao Ji, Yuanbin Wu, and Man Lan. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1237. URL `https://aclanthology.org/P19-1237`.

Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pages 462–466, 1952.

Donghwan Kim and Jeffrey A Fessler. Adaptive restart of the optimized gradient method for convex optimization. *Journal of Optimization Theory and Applications*, 178(1):240–263, 2018.

Yoon Kim, Chris Dyer, and Alexander Rush. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1228. URL `https://www.aclweb.org/anthology/P19-1228`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

Tuen Kloek and Herman K Van Dijk. Bayesian estimates of equation system parameters: an application of integration by monte carlo. *Econometrica: Journal of the Econometric Society*, pages 1–19, 1978.

Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://aclanthology.org/P10-1001.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA, October 2010. Association for Computational Linguistics. URL https://aclanthology.org/D10-1125.

Rahul G Krishnan, Simon Lacoste-Julien, and David Sontag. Barrier frank-wolfe for marginal inference. *Advances in Neural Information Processing Systems*, 28, 2015.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. *Advances in neural information processing systems*, 28, 2015.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1030. URL https://www.aclweb.org/anthology/N16-1030.

Lev Davidovich Landau and Evgenii M Lifshitz. *Statistical physics: V. 5: course of theoretical physics*. Pergamon press, 1968.

Joseph Le Roux, Antoine Rozenknop, and Mathieu Lacroix. Representation learning and dynamic programming for arc-hybrid parsing. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 238–248, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/K19-1023. URL https://www.aclweb.org/anthology/K19-1023.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Claude Lemaréchal. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer, 2001.

Ying Li, Zhenghua Li, Min Zhang, Rui Wang, Sheng Li, and Luo Si. Self-attentive biaffine dependency parsing. In *IJCAI*, pages 5067–5073, 2019.

Zuchao Li, Hai Zhao, and Kevin Parnow. Global greedy dependency parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8319–8326, 2020.

Marina Litvak, Mark Last, and Menahem Friedman. A new approach to improving multilingual summarization using a genetic algorithm. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 927–936, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://aclanthology.org/P10-1095.

Changyi Ma and Wenye Li. Sparse binary optimization for text classification via frank wolfe algorithm. In *Proceedings of the 2020 12th International Conference on Machine Learning and Computing*, pages 171–176, 2020.

Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): 313–330, 1993. URL https://www.aclweb.org/anthology/J93-2004.

André Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mário Figueiredo. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44, Cambridge, MA, October 2010. Association for Computational Linguistics. URL https://aclanthology.org/D10-1004.

André Martins, Miguel Almeida, and Noah A. Smith. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://aclanthology.org/P13-2109.

Ryan McDonald and Joakim Nivre. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230, March 2011. doi: 10.1162/coli_a_00039. URL https://aclanthology.org/J11-1007.

Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88, Trento, Italy, April 2006. Association for Computational Linguistics. URL https://aclanthology.org/E06-1011.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics. URL https://aclanthology.org/H05-1066.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

BF Mitchell, Vladimir Fedorovich Dem'yanov, and VN Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM Journal on Control*, 12(1):19–26, 1974.

Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

Alireza Mohammadshahi and James Henderson. Recursive Non-Autoregressive Graph-to-Graph Transformer for Dependency Parsing with Iterative Refinement. *Transactions of the Association for Computational Linguistics*, 9:120–138, 03 2021. ISSN 2307-387X. doi: 10.1162/tacl_a_00358. URL https://doi.org/10.1162/tacl_a_00358.

Erik Nijkamp, Mitch Hill, Tian Han, Song-Chun Zhu, and Ying Nian Wu. On the anatomy of mcmc-based maximum likelihood learning of energy-based models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5272–5280, 2020.

Noriki Nishida and Hideki Nakayama. Unsupervised discourse constituency parsing using Viterbi EM. *Transactions of the Association for Computational Linguistics*, 8:215–230, 2020. doi: 10.1162/tacl_a_00312. URL https://www.aclweb.org/anthology/2020.tacl-1.15.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160, Nancy, France, April 2003. URL https://aclanthology.org/W03-3017.

Joakim Nivre and Jens Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219853. URL https://aclanthology.org/P05-1013.

Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, 2016.

Constituency Parsing. Speech and language processing. 2009.

Fabian Pedregosa, Geoffrey Negiar, Armin Askari, and Martin Jaggi. Linearly convergent frank-wolfe with backtracking line-search. In *International Conference on Artificial Intelligence and Statistics*, pages 1–10. PMLR, 2020.

Hao Peng, Sam Thomson, and Noah A. Smith. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1186. URL `https://aclanthology.org/P17-1186`.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL `https://www.aclweb.org/anthology/D14-1162`.

Slav Petrov. Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, Los Angeles, California, June 2010. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/N10-1003`.

Slav Petrov, Leon Barrett, and Dan Klein. Non-local modeling with a mixture of PCFGs. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 14–20, New York City, June 2006. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W06-2903`.

Xian Qian and Yang Liu. Branch and bound algorithm for dependency parsing with non-local features. *Transactions of the Association for Computational Linguistics*, 1:37–48, 2013. doi: 10.1162/tacl_a_00208. URL `https://www.aclweb.org/anthology/Q13-1004`.

Marc'Aurelio Ranzato, Y-Lan Boureau, Sumit Chopra, and Yann LeCun. A unified energy-based framework for unsupervised learning. In *Artificial Intelligence and Statistics*, pages 371–379. PMLR, 2007.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=ryQu7f-RZ`.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

BTCGD Roller, C Taskar, and D Guestrin. Max-margin markov networks. *Advances in neural information processing systems*, 16:25, 2004.

Amirmohammad Rooshenas, Dongxu Zhang, Gopal Sharma, and Andrew McCallum. Search-guided, lightly-supervised training of structured prediction energy networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi 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*, pages 1–11, Cambridge, MA, October 2010. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/D10-1001`.

Itiroo Sakai. Syntax in universal translation. In *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*, 1961.

Lothar M. Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1): 1–61, 2001. ISSN 0304-3975. doi: https://doi.org/10.1016/S0304-3975(00)00406-0. URL `https://www.sciencedirect.com/science/article/pii/S0304397500004060`.

Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(2), 2013.

Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

David A. Smith and Jason Eisner. Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 145–156, Honolulu, October 2008. URL `http://cs.jhu.edu/~jason/papers/#smith-eisner-2008-bp`.

David A. Smith and Noah A. Smith. Probabilistic models of nonprojective dependency trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 132–140, Prague, Czech Republic, June 2007a. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/D07-1014`.

David A Smith and Noah A Smith. Probabilistic models of nonprojective dependency trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 132–140, 2007b.

Noah A Smith. Linguistic structure prediction. *Synthesis lectures on human language technologies*, 4(2):1–274, 2011.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014a. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014b.

Gilbert Strang, Gilbert Strang, Gilbert Strang, and Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. *Advances in neural information processing systems*, 16, 2003.

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

Alicia Tsai and Laurent El Ghaoui. Sparse optimization for unsupervised extractive summarization of long documents with the frank-wolfe algorithm. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 54–62, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.sustainlp-1.8. URL `https://aclanthology.org/2020.sustainlp-1.8`.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104, 2004.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, Yasemin Altun, and Yoram Singer. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(9), 2005.

Lifu Tu and Kevin Gimpel. Learning approximate inference networks for structured prediction. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

Lifu Tu, Tianyu Liu, and Kevin Gimpel. An Exploration of Arbitrary-Order Sequence Labeling via Energy-Based Inference Networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5569–5582, Online, November 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.449. URL `https://aclanthology.org/2020.emnlp-main.449`.

Lifu Tu, Richard Yuanzhe Pang, and Kevin Gimpel. Improving joint training of inference networks and structured prediction energy networks. In *Proceedings of the Fourth Workshop on Structured Prediction for NLP*, pages 62–73, Online, November 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.spnlp-1.8. URL https://aclanthology.org/2020.spnlp-1.8.

Lifu Tu, Richard Yuanzhe Pang, Sam Wiseman, and Kevin Gimpel. ENGINE: Energy-based inference networks for non-autoregressive machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2819–2826, Online, July 2020c. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.251. URL https://aclanthology.org/2020.acl-main.251.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wenhui Wang and Baobao Chang. Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1218. URL https://www.aclweb.org/anthology/P16-1218.

Xinyu Wang and Kewei Tu. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China, December 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.aacl-main.12.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

Michael L Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, and Andrew McCallum. Samplerank: Training factor graphs with atomic gradients. In *ICML*, 2011.

Sam Wiseman and Yoon Kim. Amortized bethe free energy minimization for learning mrfs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/dc554706afe4c72a60a25314cbaece80-Paper.pdf.

Philip Wolfe. Convergence theory in nonlinear programming. *Integer and nonlinear programming*, pages 1–36, 1970.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=HkwZSG-CZ.

Xudong Zhang, Joseph Le Roux, and Thierry Charnois. Strength in numbers: Averaging and clustering effects in mixture of experts for graph-based dependency parsing. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, pages 106–118, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.iwpt-1.11. URL https://aclanthology.org/2021.iwpt-1.11.

Yu Zhang, Zhenghua Li, and Min Zhang. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online, July 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.302. URL https://aclanthology.org/2020.acl-main.302.

Yu Zhang, Houquan Zhou, and Zhenghua Li. Fast and accurate neural CRF constituency parsing. In *Proceedings of IJCAI*, pages 4046–4053, 2020b. doi: 10.24963/ijcai.2020/560. URL https://doi.org/10.24963/ijcai.2020/560.

Yuan Zhang, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson. Steps to excellence: Simple inference with refined scoring of dependency trees. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 197–207, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1019. URL https://aclanthology.org/P14-1019.

Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL https://aclanthology.org/P11-2033.

Junbo Jake Zhao, Michaël Mathieu, and Yann LeCun. Energy-based generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=ryh9pmcee.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. A neural probabilistic structured-prediction model for transition-based dependency 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)*, pages 1213–1222, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1117. URL https://aclanthology.org/P15-1117.

Junru Zhou, Zuchao Li, and Hai Zhao. Parsing all: Syntax and semantics, dependencies and spans. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4438–4449, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.398. URL `https://www.aclweb.org/anthology/2020.findings-emnlp.398`.

Hao Zhu, Yonatan Bisk, and Graham Neubig. The return of lexical dependencies: Neural lexicalized pcfgs, 2020.

Ran Zmigrod, Tim Vieira, and Ryan Cotterell. Efficient computation of expectations under spanning tree distributions, 2020.

APPENDIX A
ENERGY-BASED MODELS

A.1 Hinge Loss Without the Outer Max

This can be seen from the derivation:

$$E(x, y; \Theta) + \max_{y' \in \mathcal{Y}} (\Delta(y, y') - E(x, y'; \Theta)) \geq E(x, y; \Theta) + \Delta(y, y) - E(x, y; \Theta)$$
$$= 0$$

Thus, we have always $E(x, y; \Theta) + \max_{y' \in \mathcal{Y}} (\Delta(y, y') - E(x, y'; \Theta)) \geq 0$ if exact inference is available. When exact inference is hard to obtain or we can have at most approximate inference, the outer max can be used to avoid negative loss values.

APPENDIX B
MIXTURE OF EXPERTS

B.1 Marginal Probability of Arc for Mixture Model

With Eq. (4.3). The marginal probability of mixture model can be written as:

$$p((h,d)|x) = \sum_{\substack{y \in \mathcal{Y}(x) \\ (h,d) \in y}} \sum_{k=1}^{K} \omega_k p_k(y|x)$$

By changing the order of sum, we can have:

$$p((h,d)|x) = \sum_{k=1}^{K} \omega_k \sum_{\substack{y \in \mathcal{Y}(x) \\ (h,d) \in y}} p_k(y|x)$$

The inner part is exactly $p_k((h,d)|x)$. Thus, we have:

$$p((h,d)|x) = \sum_{k=1}^{K} \omega_k p_k((h,d)|x)$$

B.2 Quick Gradient Analysis of Gating Network

We start from Eq. (4.6).

For mixture model with well trained experts, most of the data are equivalent for all experts, which means $p_k(y|x)$ have similar value for all experts. To see quickly why gradient approaches 0 in this case, we assume further that $p_k(y|x)$ has the same value for equivalent data. Thus, Eq. (4.6) becomes:

$$\frac{\partial L(\phi,\theta)}{\partial \phi} = \sum_{k=1}^{K} \omega_k(\phi) \frac{\partial \log \omega_k(\phi)}{\partial \phi}$$

With a little more deduction, we have:

$$\begin{aligned}
\frac{\partial L(\phi,\theta)}{\partial \phi} &= \sum_{k=1}^{K} \omega_k(\phi) \frac{1}{\omega_k(\phi)} \frac{\partial \omega_k(\phi)}{\partial \phi} \\
&= \sum_{k=1}^{K} \frac{\partial \omega_k(\phi)}{\partial \phi} \\
&= \frac{\partial \sum_{k=1}^{K} \omega_k(\phi)}{\partial \phi} \\
&= \frac{\partial 1}{\partial \phi} \\
&= 0
\end{aligned}$$

As the function is continuous w.r.t. p_k, for data which provides similar value of probability on all experts, the gradient will approaches zero. Thus, for training with Eq. (4.6), only a small part of data, which shows strong preference of particular experts, is used to train the gating network.

For training with Eq. (4.8), all the data is useful for training the gating network. In fact, the gradient of Eq. (4.8) becomes zero when:

$$\omega_k(\phi) = \frac{p_k(\theta_k)}{\sum_{k'=1}^{K} p_{k'}(\theta_{k'})}$$

Thus for data which are equivalent for all experts, a uniform weight will be learnt while for data with strong preference of particular experts, a biased weight proportional to the probability correctness on each expert can also be learnt.

B.3 Gating Network Structure, Hyper-parameters of Training

The gating network structure is similar to the structure of parse model.

Embedding Word embedding for word x_i is an concatenation of two parts: normal word embedding and CharLSTM embedding:

$$e_i = \text{emb}(x_i) \bigoplus \text{CharLSTM}(x_i)$$

when there is pre-trained embedding, the first item is the sum of word embedding calculated

by neural network, and the exterior pretrained embedding:

$$\text{emb}(x_i) = \text{WordEMB}(x_i) + \text{PreEMB}(x_i)$$

We suppose that PreEMB has the same size as WordEMB

BiLSTM The embedding vectors are then passed to 3 layers of BiLSTM, with the output at position i is noted as h_i.

Coefficient Extractor The coefficient extractor part is constructed of one layer of LSTM [Hochreiter and Schmidhuber, 1997] and one layer of MLP. The last hidden state of LSTM is passed to MLP, which compress the vector size to the number of experts in the mixture model. Two groups of coefficient extractor are used to calculate separately the weight of combination for arc and label. We note the output of MLP as $\mathbf{C} \in \mathbb{R}^K$, with:

$$\mathbf{C}_{\text{arc}} = \text{MLP}_{\text{arc}}(\text{LSTM}_{\text{arc}}(h_0, ..., h_n))$$

$$\mathbf{C}_{\text{label}} = \text{MLP}_{\text{label}}(\text{LSTM}_{\text{label}}(h_0, ..., h_n))$$

The output of MLP is passed to Softmax to calculate the weight for each expert:

$$[\omega_1, ..., \omega_K] = \text{Softmax}(\mathbf{C}_{\text{arc}})$$

$$[\omega_1^l, ..., \omega_K^l] = \text{Softmax}(\mathbf{C}_{\text{label}})$$

Model hyper-parameters of fine tuning is shown in Table B.1. We use also Adam [Reddi et al., 2018] for training, with learning rate set to $2e^{-4}$ (10 times smaller than learning rate used for training experts). The patience is set to 20 instead of the original value 100. For fine tuning, we found that best score is usually achieved in less than 20 epochs and does not increase later.

Param	**Value**	**Param**	**Value**
WordEMB size	100	Embedding dropout	0.33
CharLSTM size	50	CharLSTM dropout	0.00
BiLSTM size	400	BiLSTM dropout	0.33
LSTM_{arc} size	400	LSTM_{arc} dropout	0.00
$\text{LSTM}_{\text{label}}$ size	400	$\text{LSTM}_{\text{label}}$ dropout	0.00
MLP_{arc} size	K	MLP_{arc} dropout	0.00
$\text{MLP}_{\text{label}}$ size	K	$\text{MLP}_{\text{label}}$ dropout	0.00
Learning Rate	$\mathbf{2e^{-4}}$	β_1, β_2	0.90
Annealing	$0.75^{\frac{t}{5000}}$	Patience	**20**

Table B.1: Hyper-parameters of Fine Tuning

B.4 Implementation Differences

We implement Zhang et al. [2020a] CRF model and CRF2o model with two tiny technical differences.

The first one is that the CharLSTM [Lample et al., 2016] part in Zhang et al. [2020a] treats the beginning of the sentence `<bos>` (the special token to represent the beginning of the sentence) as five separate characters: `<,b,o,s,>`.

Our implementation treats the beginning of sentence as one special character for CharLSTM.

Another difference is that Zhang et al. [2020a] treats the lengths of every sentence as $n+2$ by considering two special tokens `<bos>` and `<eos>` (although in practice, only `<bos>` was added to every sentence). In our implementation, we keep the length of sentence as the number of words n. This is because the log probability of arc and label only considers the words in the sentence without special tokens. Thus our batch size should be a little bit higher than Zhang et al. [2020a].

One final difference is that for MBR decoding, Zhang et al. [2020a] maximizes the sum of marginal arc probability. While in our implementation of MBR, we maximize the product of marginal arc probability.

B.5 Variance Reduction on CoNLL09

We note that the label variance for FOP and SOP are quite similar that they overlap together for CoNLL09 Chinese.

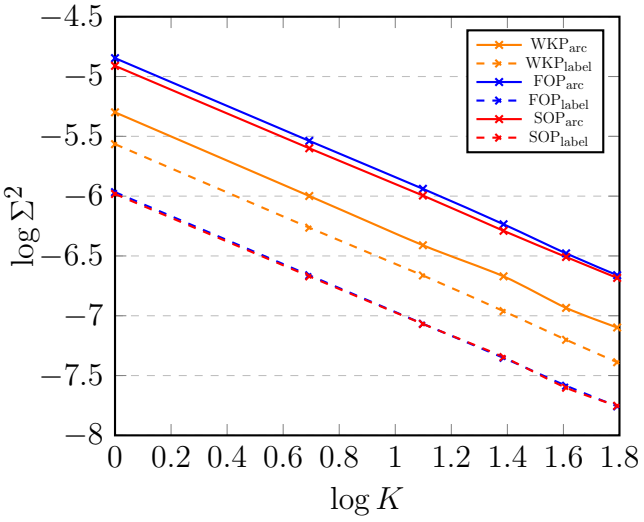

Figure B.1: Variance of System to CoNLL09 Chinese

APPENDIX C
POLYNOMIAL MODEL FOR DEPENDENCY PARSING

C.1 Hyper Parameters

Param	Value	Param	Value
WordEMB	100	WordEMB dropout	0.33
CharLSTM	50	CharLSTM dropout	0.00
PosEMB	100	PosEMB dropout	0.33
BERT Linear	100	BERT Linear dropout	0
BiLSTM	400	BiLSTM dropout	0.33
MLP_{arc}	500	LSTM_{arc} dropout	0.33
$\text{MLP}_{\text{label}}$	100	$\text{LSTM}_{\text{label}}$ dropout	0.33
$\text{MLP}_{\text{sib,gp,3O}}$	100	MLP_{arc} dropout	0.33
Learning Rate	$\mathbf{2e^{-4}}$	β_1, β_2	0.90
Annealing	$0.75^{\frac{t}{5000}}$	Patience	100

Table C.1: Hyper-parameters

Remark that when running experiments with UD, the WordEMB is reset to 300 because we use 300 dimension fasttext embedding [Mikolov et al., 2018] following [Zhang et al., 2020a, Wang and Tu, 2020].

C.2 Complete Derivations

C.2.1 Partial Derivatives

We start with the definition:

$$\frac{\partial S(y)}{\partial y_a} = \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a}$$

case a $\notin$ F: we can see that if $a \notin F$, then $\frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a} = 0$ since the expression in the numerator does not contain variable y_a.

case a $\in$ F: Now suppose that $a \in F$. Remark that F is a factor from $\mathcal{F}_k(C)$, and thus is a proper subset of arcs and consequently all arcs in F are different. By applying the rule for

product derivatives we can rewrite the partial as:

$$\frac{\partial \prod_{a' \in F} y_{a'}}{\partial y_a} = \prod_{a' \in F \backslash a} y_{a'}$$

Suppose that F is a factor of k arcs from $\mathcal{F}_k(C)$ that contains a, and that the previous equation equals 1, we have:

$$\begin{aligned}
\prod_{a' \in F \backslash a} y_{a'} = 1 &\iff y_{a'} = 1, \forall a' \in F \backslash a \\
&\iff a' \in y, \forall a' \in F \backslash a \\
&\iff F \backslash a \in \mathcal{F}_{k-1}(y)
\end{aligned}$$

Conclusion: By plugging this into the definition we have:

$$\frac{\partial S(y)}{\partial y_a} = \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ a \in F}} s_F \mathbf{1}[F \backslash a \in \mathcal{F}_{k-1}(y)]$$

C.2.2 Substitution Scores 1

We start from equation (5.1):

$$S(y) = \sum_{k=1}^{K} \sum_{F \in (\mathcal{F}_k(C) \cap \mathcal{R})} s_F \prod_{(h',d') \in F}^{k} y_{h',d'}$$

Similarly, given arc $(h, d) \in y$ we have:

$$S(y \backslash (h,d)) = \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}) \\ (h,d) \notin F}} s_F \prod_{(h',d') \in F}^{k} y_{h',d'}$$

The score difference is:

$$\begin{aligned}
&S(y) - S(y\backslash(h,d)) \\
=&\sum_{k=1}^{K} \sum_{\substack{F\in(\mathcal{F}_k(C)\cap\mathcal{R}) \\ (h,d)\in F}} s_F \prod_{(h',d')\in F}^{k} y_{h',d'} \\
=&\sum_{k=1}^{K} \sum_{\substack{F\in(\mathcal{F}_k(C)\cap\mathcal{R}) \\ (h,d)\in F}} s_F \mathbf{1}[F \in \mathcal{F}_k(y)] \\
=&\sum_{k=1}^{K} \sum_{\substack{F\in(\mathcal{F}_k(C)\cap\mathcal{R}) \\ (h,d)\in F}} s_F \mathbf{1}[F\backslash(h,d) \in \mathcal{F}_{k-1}(y)]
\end{aligned}$$

where the last line is correct since we assume we already have $(h,d) \in y$.

By using equation (5.3), we have directly:

$$S(y) - S(y\backslash(h,d)) = \frac{\partial S(y)}{\partial y_{h,d}}$$

which is

$$S(y) = \frac{\partial S(y)}{\partial y_{h,d}} + S(y\backslash(h,d))$$

C.2.3 Substitution Scores 2

First, note that the set of arc $y\backslash(h,d)$ is the same as $y[h \to h',d]\backslash(h'd)$. This is because $y[h \to h',d]$ is constructed by substituting arc $(h,d) \in y$ with arc (h',d). The other arcs are unchanged. Thus we have:

$$S(y[h \to h',d]\backslash(h',d)) = S(y\backslash(h,d))$$

Secondly, consider the condition:

$$(h',d) \in F, F\backslash(h',d) \in \mathcal{F}_{k-1}(y[h \to h',d])$$

Remark that $F = \{(h_1,d_1),(h_2,d_2),...,(h_k,d_k)\}$ being a proper subset of arcs is required to satisfy: $\forall i \neq j$, $d_i \neq d_j$. Thus $F\backslash(h',d)$ has no arc for column d. In this case, we can write

the previous condition as:

$$(h', d) \in F, F \backslash (h', d) \in \mathcal{F}_{k-1}(y)$$

since y and $y[h \rightarrow h', d]$ only differ in column d.

By using equation (5.3), we have:

$$\begin{aligned}
&\frac{\partial S(y[h \rightarrow h', d])}{\partial y_{h',d}} \\
&= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ (h',d) \in F}} \\
&\qquad s_F \mathbf{1}[F \backslash (h', d) \in \mathcal{F}_{k-1}(y[h \rightarrow h', d]) \\
&= \sum_{k=1}^{K} \sum_{\substack{F \in (\mathcal{F}_k(C) \cap \mathcal{R}), \\ (h',d) \in F}} s_F \mathbf{1}[F \backslash (h', d) \in \mathcal{F}_{k-1}(y) \\
&= \frac{\partial S(y)}{\partial y_{h',d}}
\end{aligned}$$

To conclude, we have:

$$\begin{aligned}
&S(y[h \rightarrow h', d]) \\
&= \frac{\partial S(y[h \rightarrow h', d])}{\partial y_{h',d}} + S(y[h \rightarrow h', d] \backslash (h', d)) \\
&= \frac{\partial S(y)}{\partial y_{h',d}} + S(y \backslash (h', d))
\end{aligned}$$

The first equation is a direct usage of equation (5.4) and the second equation comes from the previous proof.

C.2.4 First-order Linearization

We want to compute for all word positions d the highest scoring head:

$$
\begin{aligned}
&\operatorname*{argmax}_{h'} S(y[h \to h', d]) \\
\approx &\operatorname*{argmax}_{h'} S(y) + (y[h \to h', d] - y)^\top \nabla S(y) \\
= &\operatorname*{argmax}_{h'} S(y) + \frac{\partial S(y)}{y_{h'd}} - \frac{\partial S(y)}{y_{hd}} \\
= &\operatorname*{argmax}_{h'} \frac{\partial S(y)}{y_{h'd}}
\end{aligned}
$$

We go from first to second line by first-order Taylor approximation. Transition from second to third line is based on the fact that $y[h \to h', d]$ differs from y by only two arcs, the addition of (h', d) and the removal of (h, d) so the inner product can be expressed as a difference of two partial derivatives. We go from third to fourth line by noticing that only one term depends on h' hence we can simplify the argmax.

C.2.5 Approximate Marginal Estimation

$\hat{y}$ is the highest-scoring parse and contains arc (g, d). We write $s_{hd} = \frac{\partial S(\hat{y})}{\partial y_{hd}}$ for all arc (h, d). We recall from previous section that first-order Taylor approximation gives: $S(y[g \to h, d]) \approx S(\hat{y}) + s_{hd} - s_{gd}$.

$$
\begin{aligned}
p\big((h, d)|x^*\big) &= \frac{p(\hat{y}[g \to h, d])}{\sum_{h'} p(\hat{y}[g \to h', d])} \\
&= \frac{Z^{-1} \exp(S(\hat{y}[g \to h, d]))}{\sum_{h'} Z^{-1} \exp(S(\hat{y}[g \to h', d]))} \\
&= \frac{\exp(S(\hat{y}[g \to h, d]))}{\sum_{h'} \exp(S(\hat{y}[g \to h', d]))} \\
&\approx \frac{\exp(S(\hat{y}) + s_{hd} - s_{gd})}{\sum_{h'} \exp(S(\hat{y}) + s_{h'd} - s_{gd})} \\
&= \frac{\exp(S(\hat{y}) - s_{gd}) \exp(s_{hd})}{\exp(S(\hat{y}) - s_{gd}) \sum_{h'} \exp(s_{h'd})} \\
&= \frac{\exp(s_{hd})}{\sum_{h'} \exp(s_{h'd})}
\end{aligned}
$$

C.3 Tensor Factorization for Third-Order Models

For a third order model, a tensor $W \in \mathbb{R}^{n^6}$ should be used to calculate the score of $F = \{(h_1, d_1), (h_2, d_2), (h_3, d_3)\}$:

$$s_F = v_{h_3}^T v_{h_2}^T v_{h_1}^T W v_{d_1} v_{d_2} v_{d_3}$$

with v_{h_i}, v_{d_i} the feature vector of head and modifier words.

To reduce the memory cost, we simulate the previous calculation with three tensors of biaffine and one tensor of triaffine. The score can be calculated as:

$$\begin{aligned} l_1 &= v_{h_1} \circ W_{biaffine}^{(1)} v_{d_1} \\ l_2 &= v_{h_2} \circ W_{biaffine}^{(2)} v_{d_2} \\ l_3 &= v_{h_3} \circ W_{biaffine}^{(3)} v_{d_3} \\ s_F &= l_3^T l_2^T W_{triaffine} l_1 \end{aligned}$$

with $W_{biaffine}^i \in \mathbb{R}^{n^2}$ the tensor of biaffine and $W_{triaffine} \in \mathbb{R}^{n^3}$ the tensor of triaffine, $\circ$ represents the Hadamard product (element-wise product of vector).

APPENDIX D
NON-LINEAR MODEL FOR DEPENDENCY PARSING

D.1 Projection

The equivalence can be seen from:

$$\begin{aligned}
\underset{y' \in \mathcal{Y}}{\operatorname{argmin}} ||y' - y||_1 &\iff \underset{y' \in \mathcal{Y}}{\operatorname{argmin}} \langle y', 1 - y \rangle + \langle 1 - y', y \rangle \\
&\iff \underset{y' \in \mathcal{Y}}{\operatorname{argmin}} -2\langle y', y \rangle \\
&\iff \underset{y' \in \mathcal{Y}}{\operatorname{argmax}} \langle y', y \rangle
\end{aligned}$$

QED.

D.2 Gradient of KL

$$\begin{aligned}
\frac{d\overrightarrow{L^{\mathrm{KL}}_G}(\Phi)}{d\Phi} &= \frac{\partial}{\partial \Phi} \sum_{y \in \mathcal{Y}} q(y;\Phi) \log \frac{q(y;\Phi)}{p(y;\Theta)} \\
&= \sum_{y \in \mathcal{Y}} (1 + \log \frac{q(y;\Phi)}{p(y;\Theta)}) \frac{\partial q(y;\Phi)}{\partial \Phi} \\
&= \sum_{y \in \mathcal{Y}} \frac{q(y;\Phi)}{q(y;\Phi)} (1 + \log \frac{q(y;\Phi)}{p(y;\Theta)}) \frac{\partial q(y;\Phi)}{\partial \Phi} \\
&= \mathbb{E}_{y \sim q(y;\Phi)} (1 + \log q(y;\Phi) - \log p(y;\Theta)) \frac{\partial \log q(y;\Phi)}{\partial \Phi}
\end{aligned}$$

In third line, we add the naive term $\frac{q(y;\Theta)}{q(y;\Theta)}$ to represent the calculation of the gradient as an expectation to the easy to calculate and sample distribution $q(y;\Phi)$.

Similarly we have:

$$\begin{aligned}
\frac{d\overleftarrow{L^{\mathrm{KL}}_G}(\Phi)}{d\Phi} &= \frac{\partial}{\partial \Phi} \sum_{y \in \mathcal{Y}} p(y;\Theta) \log \frac{p(y;\Theta)}{q(y;\Phi)} \\
&= \sum_{y \in \mathcal{Y}} -p(y;\Theta) \frac{\partial \log q(y;\Phi)}{\partial \Phi} \\
&= \mathbb{E}_{y \sim q(y;\Phi)} - \frac{p(y;\Theta)}{q(y;\Phi)} \frac{\partial \log q(y;\Phi)}{\partial \Phi}
\end{aligned}$$

D.3 Pairwise FW for Dependency Parsing

Algorithm 17: Pairwise Frank-Wolfe Algorithm for Dependency Parsing, with Backtracking Line-Search, Restart, Early Stopping and Projection

Initialization: $y^{(0)} \in \mathcal{Y}$, $S = \{y^{(0)}\}$, $\alpha = \{\alpha_v = 1 | v \in S\}$, $\epsilon, \delta > 0$, $L := L_0 > 0$, RESTART = False;
for $t \leftarrow 0$ **to** T **do**
  Let $s_t := DA(\nabla S(y^{(t)}))$ and $d_t^{\mathrm{F}} := s_t - y^{(t)}$;
  if $g_t^F := \langle d_t^F, \nabla S(y^{(t)}) \rangle \leq \epsilon$ **then**
    Projection: $y^* := \mathrm{DA}(y^{(t)})$;
    Return $y^*, S(y^*)$;
  end
  Let $d_t := d_t^{\mathrm{F}}$, $g_t := g_t^{\mathrm{F}}$, $\gamma_{\max} = 1$;
  Backtracking Line-Search: $\gamma_t, L = \mathrm{Backtracking}(S, d_t, y^{(t)}, g_t, L, \gamma_{\max})$;
  if $\gamma_t \leq \delta$ **then**
    if *RESTART* **then**
      Projection: $y^* := \mathrm{DP}(y^{(t)})$;
      Return $y^*, S(y^*)$;
    else
      Let $L := L_0$, RESTART = True;
    end
  else
    Let RESTART = False;
  end
  if $s_t \in S$ **then**
    Update $\alpha_{s_t} := \alpha_{s_t} + \gamma_t$ and $\alpha_{v_t} := \alpha_{v_t} - \gamma_t$;
  else
    Update $S := S \cup \{s_t\}$;
    Update $\alpha := \alpha \cup \{\alpha_{s_t} := \gamma_t\}$ and $\alpha_{v_t} := \alpha_{v_t} - \gamma_t\}$;
  end
  Pop $v \in S, \alpha_v \in \alpha$ with $\alpha_v = 0$;
  Update $y^{(t+1)} := y^{(t)} + \gamma_t d_t$;
end
Projection: $y^* := \mathrm{DA}(y^{(T+1)})$;
Return $y^*, S(y^*)$;

D.4 Estimation of Log Probability

$$\begin{aligned}
\log p(y;\Theta) &= \log \frac{p(y;\Theta)}{1} \\
&= \log \frac{p(y;\Theta)}{\sum_{y'\in\mathcal{Y}} p(y';\Theta)} \\
&= \log \frac{p(y;\Theta)}{\sum_{y'\in\mathcal{Y}} \frac{p(y';\Phi)}{p(y';\Phi)} p(y';\Theta)} \\
&= \log \frac{p(y;\Theta)}{\mathbb{E}_{y'\sim q(y;\Phi)} \frac{p(y';\Theta)}{p(y';\Phi)}} \\
&= \log \frac{\exp(S(y;\Theta))}{\mathbb{E}_{y'\sim q(y;\Phi)} \frac{\exp(S(y';\Theta))}{p(y';\Phi)}} \\
&= S(y;\Theta) - \log \mathbb{E}_{y'\sim q(y;\Phi)} \frac{\exp(S(y';\Theta))}{q(y;\Phi)}
\end{aligned}$$

D.5 Non-Exact Approximation

Suppose $q(y;\Phi)$ equals $p(y;\Theta), \forall y \in \mathcal{Y}$, this indicates that:

$$S(y;\Phi) = S(y;\Theta) + C, \forall y \in \mathcal{Y}$$

where C is a constant.

Since $S(y;\Phi)$ is a linear function, we can rewrite the previous equation as

$$\langle W(\Phi), y\rangle = S(y;\Theta) + C, \forall y \in \mathcal{Y}$$

where $W(\Phi) \in \mathbb{R}^{n^2}$. This indicates that the rank of the matrix $\{y^{(1)}, ..., y^{|\mathcal{Y}|}\}$ by stacking $y \in \mathcal{Y}$ together is at most n^2 while we have $|\mathcal{Y}|$ equations. Note that $|\mathcal{Y}|$ increases exponentially with n. Thus for sufficiently long sentences, we cannot guarantee there exists a solution [Strang et al., 1993].