AN ABSTRACT OF THE DISSERTATION OF

Kai Zhao for the degree of Doctor of Philosophy in Computer Science presented on May 30, 2017.

Title: Structured Learning with Latent Variables: Theory and Algorithms

Abstract approved: _____

Liang Huang

Most tasks in natural language processing (NLP) try to map structured input (e.g., sentence or word sequence) to some form of structured output (tag sequence, parse tree, semantic graph, translated/paraphrased/compressed sentence), a problem known as "structured prediction". While various learning algorithms such as the perceptron, maximum entropy, and expectation-maximization have been extended to the structured setting (and thus applicable to NLP problems), directly applying them as is to NLP tasks remains challenging for the following reasons. First, the prohibitively large search space in NLP makes exact search intractable, and in practice inexact search methods like beam search are routinely used instead. Second, the output structures are usually partially, rather than completely, annotated, which requires structured latent variables. However, the introduction of inexact search and latent components violates some key theoretical properties (such as convergence) of conventional structured learning algorithms, and requires us to develop new algorithms suitable for scalable structured learning with latent variables. In this thesis, we first investigate new theoretical properties for these structured learning algorithms with inexact search and latent variables, and then also demonstrate that structured learning with latent variables is a powerful modeling tool for many NLP tasks with less strict annotation requirements, and can be generalized to neural models.

Structured Learning with Latent Variables: Theory and Algorithms

by

Kai Zhao

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented May 30, 2017
Commencement June 2017

Doctor of Philosophy dissertation of Kai Zhao presented on May 30, 2017.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Kai Zhao, Author

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST OF TABLES (Continued)

Table Page

LIST OF ALGORITHMS

Chapter 1: Introduction

Among the various natural language processing (NLP) problems, one general but fundamental type of problems is *structured prediction*. Structured prediction problems map structured input (sentences) to structured output (tag sequences, parse trees, semantic graphs, translated/paraphrased/compressed sentences). Many classical NLP tasks like Part-Of-Speech tagging, constituent/dependency parsing, and machine translation belong to it. Structured prediction problems, together with their learning algorithms, are of central importance in NLP area.

1.1 Typical Structured Prediction Problems

Most structured prediction problems in NLP area takes a sentence, i.e., a sequence of words, as input, but their output structures vary based on different problems, which means different time complexities in searching for the optimal output results. In this section we will have a brief survey of the different structured prediction problems.

The first type of structured prediction problem is *sequence labeling* problem. As its name suggests, this type of problem tries to assign a label for each word in the input sequence, which means its output is also a sequence. A typical sequence labeling problem is Part-Of-Speech (POS) tagging problem, which labels each word in the input sentence with a label from a predefined label set that represents the functionality of the word (i.e., verb, noun, pronoun, etc.) in the sentence. The determination of the labeling for one word depends on the syntactic role of the word in the sentence, which means in theory the whole sentence need to be considered in assigning the POS tag. However, in practice, for efficiency consideration, only the segment near the word is considered, which we will discuss later in Section 2.1. An example of POS tagging is demonstrated in Figure 1.1.

Another example of sequence labeling task is *Named Entity Recognition* (NER), which labels those words or phrases in the sentences that are mentions of some entities of certain predefined types. Since NER labels spans of the input sequence, it usually labels the beginnings and continuations of the mentions. Figure 1.1 also show an example of NER

	Rolls-Royce	Motor	Cars	Inc.	said
POS tagging	NNP	NNP	NNP	NNP	VBD
NER	B-ORG	I-ORG	I-ORG	I-ORG	-
	it	expects	its	U.S.	sales
POS tagging	PRP	VBZ	PRP$	NNP	NNS
NER	-	-	-	B-LOC	-
	to	remain	steady	at	about
POS tagging	TO	VB	JJ	IN	IN
NER	-	-	-	-	-
	1,200	cars	in	1990	.
POS tagging	CD	NNS	IN	CD	.
NER	-	-	-	B-DAT	-

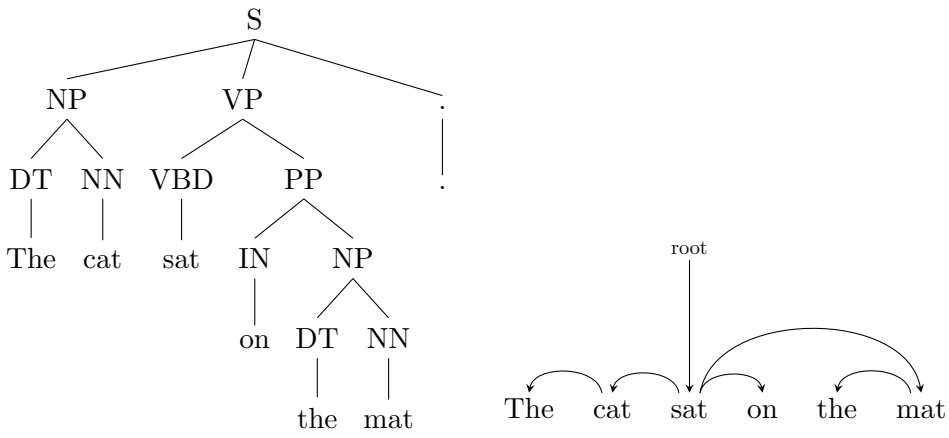
Figure 1.1: Examples of sequence labeling problems: POS tagging and Named Entity Recognition (NER). The input is an English sentence (sequence), the output is an sequence of annotations for each word in the input sentence. The POS tagging annotations are syntactic part-of-speech tags. The NER annotations are labels of the associated entity types, where B- and I- mean the beginning of an entity mention, and the continuation of an entity mention, respectively.

labeling. In this example, there are 4 types of entities recognized: person (PER), location (LOC), organization (ORG), and DATE (DAT).

One step further from the sequence labeling problem we have the classical *parsing* problem where the input is still a sentence, but the output is a parse tree. Parsing problem usually deals with two types of parses: *constituent parsing*, or syntactic parsing, and *dependency parsing*.

Constituent parsing parses the input sentences to a syntax tree following a given context-free grammar. An example parse tree is shown in Figure 1.2 (a), based on the following context-free grammar:

$$
\begin{aligned}
&\text{S} \rightarrow \text{NP VP} \\
&\text{VP} \rightarrow \text{VB PP} \\
&\text{NP} \rightarrow \text{DT NN} \\
&\ldots \\
&\text{NN} \rightarrow \text{cat} \\
&\text{VB} \rightarrow \text{sit} \\
&\ldots
\end{aligned}
$$



(a) constituent parsing (b) unlabeled dependency parsing

Figure 1.2: Examples of the parsing problem.

Constituent parsing is one of the most important core NLP tasks since its outputs are very useful in linguistic analysis. There are various parsing similar to constituent parsing, but defined for different grammars, such as tree-adjoining grammar (TAG) [24], and combinatory categorial grammar (CCG) [1].

Another type of parsing different from previous parsing methods where the parsing rules, i.e., the grammar, is not explicitly defined is dependency parsing. In dependency parsing, a tree edge points from a tail word to a head word, indicating that the the tail word modifies the head word. One tail word can only modify on one head word, while one head word can be the modified by many tail words. Figure 1.2 (b) shows an example of dependency parsing for sentence "The cat sat on the mat.".

Both constituent parsing and dependency parsing are defined in the syntactic domain, i.e., they convey some syntactic information in the tree structure. Corresponding to the syntactic parsing, in the semantic domain, the meaning of a natural language sentence can also be represented by structures of either trees like lambda expressions coupled with CCG [64], lambda dependency-based compositional semantics (λ-DCS) [32], or graphs like abstract meaning representation (AMR) [4]. The task to derive the semantic parse tree is called *semantic parsing*. Figure 1.3 shows an example of the semantic parse tree with meaning representation of first-order logic lambda expression. In this representation, the composition of sub lambda expressions still forms a tree, which is the

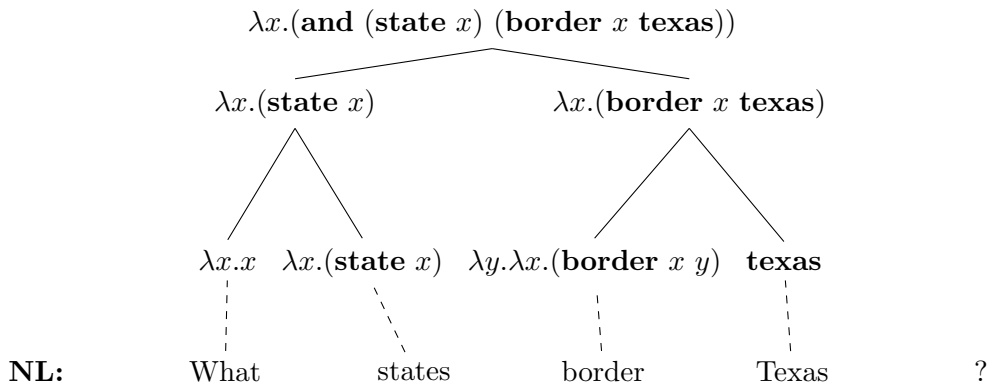


Figure 1.3: An example of semantic parsing. The natural language sentence is parsed into a semantic parse tree which has the meaning representation as its root. The meaning representation is a lambda expression of first-order logic. The dashed lines represent the grounding from natural language phrases to predicates/expressions in the meaning representation domain.

derivation tree. The searching for the correct derivation tree is similar to the syntactic parsing problem, and can either be guided by CCG, or follow the strategy of dependency parsing.

However, there is one key component that differs semantic parsing from syntactic parsing that in semantic parsing, how a natural language phrase corresponds to a first-order logic predicate in the knowledge base is not clear during the decoding time. For example, in Figure 1.3, the word "state" may mean the country United States, or a state like Oregon. This introduces extra ambiguity in the structured prediction problem, as well as in the structured learning problem where these correspondences need to be learned.

Machine Translation task is, to some extent, similar to semantic parsing task in that it also converts the input sentence to another structure that represents the same meaning. In semantic parsing, this structure can be a sequence of lambda expression or a graph, while in machine translation this structure is another sentence in a different language.

This shared property suggests that there are many common aspects between semantic parsing and machine translation.

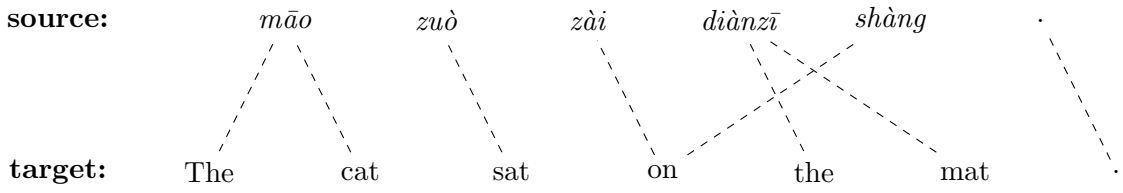


Figure 1.4: An example of the translation problem. The source sentence (sequence of words) in Chinese is translated to a target sentence (sequence of words) in English. The dashed lines represent the corresponding translation of each source word in English, which is also known as alignment. The output sequence corresponds to the input sequence, but not in the one-to-one mapping manner as in sequence labeling examples in Figure 1.1. Words deletion and reordering are necessary in translation.

1. In machine translation, how a phrase in source language can be translated to a phrase in target language, i.e., the translation rule, is unknown, and needs to be learned, similar to how a natural language phrase can be grounded to a first-order logic predicates is learned in semantic parsing.

2. Given the translation rules, how to compose these rules to translate the source sentence to target language is just another search for the correct translation derivation, which is similar to searching for the derivation tree in semantic parsing.

Figure 1.4 shows an example of machine translation where the translation rules are explicitly labeled by dashed lines.

The last task we discuss, *natural language inference*, which is also known as recognizing textual entailment (RTE), is different from all previous tasks in that its input is still structured, but its output is a label of whether the input hypothesis sentence can be inferred from the input premise sentence. However, natural language inference can still be viewed as a structured prediction problem not only because of its structured input, but also because of the structured internal inference to determine the final output label.

In practice, the sentence entailment task usually involves more than a single entailment relation. For example, in the Stanford Natural Language Inference dataset [8] there are three relations (entailment, contradiction, unrelated). In Natural Logic [34] there is a refined set of seven relations (equivalence, forward entailment, reverse entailment, negation, alternation, cover, independence).

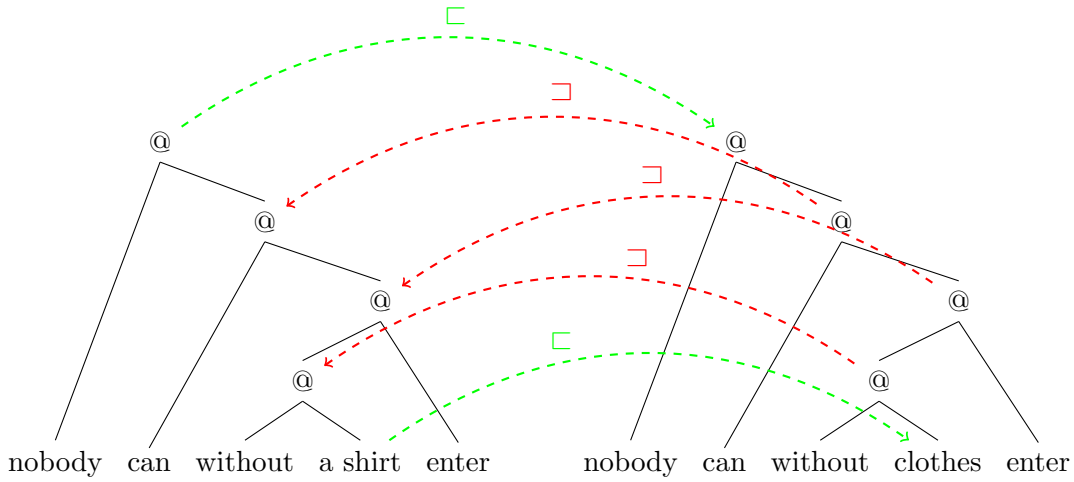


Figure 1.5: An example of sentence entailment [34]. The dashed lines represent the node-wise entailment relations between nodes of the two trees. $a \sqsupset b = b \sqsubset a$ means b can entail a. Note that for this task, although the output label is not structured, the internal entailment inference is structured.

Figure 1.5 shows a simple example of sentence entailment, in which the premise sentence, "nobody can enter without a shirt", entails the hypothesis sentence "nobody can enter without clothes". This entailment is determined by aligning the syntax trees of each sentence and identifying the entailment relation node-by-node with special considerations about the scope of negation.

1.2 Challenges in Structured Prediction and its Learning Algorithms

The structured search space of structured prediction casts new challenges for both the prediction algorithm and the learning algorithm.

In the perspective of searching for the correct output structure, the number of possible derivations is exponential to the size of the input, due to the combinatorial explosion property of the structured labels. As a result, brute-force searching for the optimal output is not affordable in time complexity. Two approaches are widely used to ease this problem:

1. Many structured prediction problems either have internal overlapping structures, or

can be approximated by simplified models with overlapping structures. Dynamic programming technique is often used to leverage this overlapping structure by decomposing the problem into small overlapping subproblems. A typical example of this approach is the Hidden Markov Model (HMM) that makes the Markov assumption for the POS tagging task that the probability of one tagging decision approximately only depends on the immediately preceding one or two word tag pairs. This approximation reduces the search time for the output sequence from exponential to polynomial. (Section 2.1)

2. Even with dynamic programming, the reduced search time is usually high-order polynomial that is not affordable in practice. To overcome this scalability issue, a typical choice is to resort to approximated search like beam search. Beam search can significantly reduce the search time, but it can also highly sacrifice the accuracy, especially when the search parameters are not aware of the approximation in the search algorithm.

In the perspective of statistical learning, to optimize the model for a structured prediction problem can be viewed as a special case of the multi-class classification problem. But, instead of the typical multi-class classification with a (small) fixed number of output classes, there are exponentially many output classes in structured prediction. Exact (brute-force) search for the output class, as mentioned above, is no longer practical, and inexact (approximated) search is usually the default choice.

Many conventional machine learning algorithms have their corresponding structured variants for structured prediction tasks. For example, Perceptron [46] becomes Structured Perceptron [13]; logistic regression becomes Conditional Random Field (CRF) [31]; Support Vector Machine (SVM) [55] becomes structured SVM [54] and Max-Margin Markov Networks (M^3N) [52]. Furthermore, neural network models also evolve to its structured version like neural CRF [17, 18].

As we will discuss in Chapter 3, simply changing the searching algorithm for conventional machine learning algorithms is not enough, since the model parameters learned are unaware of the change and could lead to non-optimal output class in the approximated setting.

1.3 Structured Learning with Natural Annotations

There is another challenge we need to mention for structured learning problem, which is not inherently a challenge from the structured learning algorithms, but a challenge for better training.

Nowadays, most popular structured learning algorithms rely on large training corpora to achieve state-of-the-art performance. The discriminative training algorithms like structured Perceptron and CRF all are trained on large training corpora to learn useful complex features to discriminate bad examples from correct ones. The neural network models also need on large training corpora to overcome the overfitting problem caused by the huge parameter space.

The availability of large training corpora, especially those that require careful human annotation is always an issue for structured learning. For example, for parsing task, annotating natural language sentences with parse trees requires prior knowledge in linguistics, which is highly non-trivial even for widely used languages like English. Another example is for the machine translation task, where to annotate one translation derivation, i.e., what are the translation rules being used, and how the rules are applied, is hardly possible.

Several approaches address this issue. One possible solution, Uptraining [43], is to manually annotate a small portion of the training data, and then train a structured learning model over this small portion of data, which is being used to automatically annotate a large portion of unannotated data. However, mistakes of the model from the small dataset can contaminate the automatic annotation, and to select the automatic annotations that are confident becomes of great importance [58]. Another approach is to apply domain adaptation method [15] to a structured learning model that is trained on an annotated corpus in some domain and then convert it to another domain that lacks annotated data. However, designing the conversion procedure is not trivial and usually requires domain-specific knowledge.

Here we will discuss a different approach: instead of relying on data annotated by professionals, we lower the required standard of the annotation so that we only require weak annotation that even unprofessional people can annotate the training data. We call this weak annotation *natural annotation*.

Note that this natural annotation idea is not new in literature. Many structured

learning algorithms exploit weakly annotated corpus and achieve accurate models.

1. One typical example is the machine translation task, where the annotation is parallel sentence pairs of a source language and a target language, rather than annotation of the translation rules and how these rules are composed to form the translation derivation. The parallel texts are usually easy to find. For example, the EUROPARL corpus consists of documents from the Europe Parliament in 21 European languages [25].

2. Semantic parsing task also uses natural annotations so that the structured learning model can learn from question answer pairs [28, 5]. In this case, correct semantic parses will derive correct answers and, thus, will be rewarded, and incorrect parses will be penalized.

3. Similarly, the entailment task can benefit from natural annotations. Note that in our example (Figure 1.5), to annotate the tree node correspondence and node-wise entailment relation is time consuming and impractical. Annotating the sentence-level entailment relation is a far easier task, and we would like to learn our structured model from this weak annotation.

With natural annotations training corpora are significantly easier to collect. Comparing with the standard fully annotated corpus for constituent parsing, Penn Treebank [35], which contains $\sim 40k$ tree parse annotation, standard machine translation corpora usually contain millions of sentence pairs.

The common property of those structured learning tasks from natural annotations is that, instead of knowing the whole searching derivation from the *full* annotation, we can only observe *partial* weak annotation. This means that, during our structured learning, we need to hallucinate the correct searching derivations that agree with the natural annotations and use these derivations as the target to reward. In other words, these derivations are hidden, or *latent*, to our models. This technique will be discussed in Chapter 4.

Just like the input and output of structured prediction problems, these latent components are structured, and searching over the possible structured space is time consuming, which introduces further difficulties for this kind of *structured learning problem with latent variables*.

In later chapters, we will first establish a mathematical framework for structured prediction and structured learning, and then discuss how introducing latent variables will affect the properties of the structured learning models both theoretically and practically.

Chapter 2: Structured Prediction Problems

In this chapter we first establish the mathematical formalization for the structured prediction problem based on a simple sequence labeling task: Part-Of-Speech (POS) tagging. Then we will see how this framework adapts to different structured prediction problems.

2.1 Mathematical Formalization for Sequence Labeling

We start with the very simple sequence labeling task, POS tagging, to establish the mathematical formalization. This formalization will be used in later sections to analyze the properties of structured prediction and structured learning. An example of POS tagging task is illustrated in Figure 1.1.

For sequence labeling task, the input

$$x = (x_1 x_2 \dots x_m) \in \mathcal{X}$$

is a sequence of words of length m. The output is also a sequence of labels of length m:

$$y = (y_1 y_2 \dots y_m) \in \mathcal{Y},$$

where each $y_i \in \mathcal{T}$ is assigned label for word x_i, and $\mathcal{T}$ is the set of all possible labels tags. In the POS tagging task, $\mathcal{T}$ is the set of all POS tags. We also define $\mathcal{Y}(x)$ to be the set of all possible sequences of labels for input sentence x.

The structured prediction problems we are interested in is to find the best output sequence y^* for a given input sentence x, under some scoring function $F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} F(x, y). \tag{2.1}$$

Usually the scoring function $F(\cdot, \cdot)$ is modeled by some statistical model with parameters $\mathbf{w}$. So F can also be written as $F_{\mathbf{w}}$. This problem is called the **inference problem** since the goal is to infer the best label sequence y^* for the input x. It is also called the

decoding problem since it can be viewed as to find the best interpretation y^* for the coded sequence x.

As we mentioned in Chapter 1.2, solving the decoding problem for even the simple POS tagging problem has high time complexity. Note that for each label y_i in the label sequence, the number of possible choices is equal to the size of all possible POS tags, $|\mathcal{T}|$. The number of all possible label sequences $|\mathcal{Y}(x)| = |\mathcal{T}|^m$, and a linear search over all possible sequences takes $O(|\mathcal{T}|^m)$ time, which is not affordable even for a small m.

To address this issue, two strategies are usually applied: dynamic programming and approximated search.

2.1.1 Exact Inference: Viterbi Algorithm

The technique of dynamic programming relies on the properties of the scoring function $F_{\mathbf{w}}$ in Equation 2.1.

In practice, the scoring function $F_{\mathbf{w}}$ is usually defined to be *decomposable*:

$$F_{\mathbf{w}}(x, y_1 y_2 \dots y_{i-1} y_i) = F_{\mathbf{w}}(x, y_1 y_2 \dots y_{i-1}) + f_{\mathbf{w}}(y_i; x, y_1 y_2 \dots y_{i-1}),$$

where we abuse the notation so that $F_{\mathbf{w}}$ can be applied to partial sequence, and the stepwise score function $f_{\mathbf{w}}(y_i; x, y_1 y_2 \dots y_{i-1})$ evaluates the score of assigning label y_i to word x_i given all the previous label assignment history $y_1 y_2 \dots y_{i-1}$.

To simplify the calculation, we usually assume the label assignment is a *Markov process*, i.e., the assignment of label y_i only depends on a label history of fixed length k, $y_{i-k+1} \dots y_i$. This means that the calculation of the stepwise scoring function only depends on the local structure near y_i:

$$f_{\mathbf{w}}(y_i; x, y_1 y_2 \dots y_{i-1}) = f_{\mathbf{w}}(y_i; x, y_{i-k+1} \dots y_{i-1}).$$

Under this assumption, the scoring function can be decomposed to be the sum over local partial sequences:

$$F_{\mathbf{w}}(x, y) = \sum_{i=1}^{m} f_{\mathbf{w}}(y_i; x, y_{i-k+1} \dots y_{i-1})$$

The dynamic programming algorithm relies on this decomposability to avoid unnec-

essary calculation. Let $G_{\mathbf{w}}(x, y_{i-k+2} \dots y_i)$ be the maximal score of a label sequence of length i for sentence x, which ends with a sequence $y_{i-k+2} \dots y_i$ of length $k-1$. $G_{\mathbf{w}}$ can be calculated recursively:

$$\begin{aligned} G_{\mathbf{w}}(x, y_{i-k+2} \dots y_i) &= \max_{y_1,\dots,y_{i-k}} \sum_{j=1}^{i} f_{\mathbf{w}}(y_j; x, y_{j-k+1} \dots y_{j-1}) \\ &= \max_{y_{i-k+1}} G_{\mathbf{w}}(x, y_{i-k+1} \dots y_{i-1}) + f_{\mathbf{w}}(y_i; x, y_{i-k+1}, \dots, y_{i-1}) \end{aligned}$$

Similarly we can have the recursive calculation of Equation 2.1:

$$\begin{aligned} \max_{y \in \mathcal{Y}(x)} F_{\mathbf{w}}(x, y) &= \max_{y \in \mathcal{Y}(x)} \sum_{i=1}^{m} f_{\mathbf{w}}(y_i; x, y_{i-k+1} \dots y_{i-1}) \\ &= \max_{y \in \mathcal{Y}(x)} G_{\mathbf{w}}(x, y_{m-k+2} \dots y_m) \\ &= \max_{y_{m-k+2},\dots,y_m} G(x, y_{m-k+2} \dots y_m) \\ &= \max_{y_{m-k+1},\dots,y_m} G(x, y_{m-k+1} \dots y_{m-1}) + f_{\mathbf{w}}(y_m; x, y_{m-k+1} \dots y_{m-1}) \end{aligned}$$

This recursive expansion indicates that the maximal score over all possible label sequences can be calculated in a dynamic programming fashion: at step i, $G_{\mathbf{w}}(x, y_{i-k+2} \dots y_i)$ can be calculated precisely by enumerating over all $G_{\mathbf{w}}(x, y_{i-k+1} \dots y_{i-1})$ and try to append y_i, which needs $O(|\mathcal{T}|)$ time. The total time needed for calculating the optimal label sequence takes $O(m|\mathcal{T}|^k)$ time. This dynamic programming algorithm for inference the output structure is called *Viterbi Algorithm*.

2.1.2 Inexact Inference: Beam Search

Viterbi algorithm decreases the time complexity of the inference problem from $O(|\mathcal{T}|^m)$ to $O(m|\mathcal{T}|^k)$ without affecting the accuracy of the searching under the Markov assumption. However, $O(m|\mathcal{T}|^k)$ is still not fast enough if the label set $\mathcal{T}$ is big.

Further speedup can be achieved by sacrificing the accuracy of the searching, i.e., *approximated search*. One widely used approximated search algorithm is *beam search*.

In beam search, at step i to find the candidate labels for y_i, only top b values of

$G_{\mathbf{w}}(x, y_{i-k+1} \dots y_{i-1})$ are kept and further expanded to $G_{\mathbf{w}}(x, y_{i-k+2} \dots y_i)$.

More formally, we can define the beam search recursively:

$$\begin{aligned} &\mathbf{B}_0(x; \mathbf{w}) = \{\epsilon\}, \\ &\mathbf{B}_i(x; \mathbf{w}) = \mathbf{top}^b(\{y_{i-k+2} \dots y_i | y_i \in \mathcal{T}, y_{i-k+1} \dots y_{i-1} \in \mathbf{B}_{i-1}(x; \mathbf{w})\}), \end{aligned} \tag{2.2}$$

where $\mathbf{top}^b$ takes the top b candidates of form $y_{i-k+2} \dots y_i$, ranked by the scoring function $G_{\mathbf{w}}$ over a prefix of the label sequence.

With beam search, the time complexity for the inference problem for POS tagging is $O(mb|\mathcal{T}|)$, i.e., linear to the length of the input sentence, which is sufficiently fast in practice.

2.2 Discussion of General Structured Prediction Problems

The above formalization in Chapter 2.1 can easily be adapted to other structured prediction problems. Here we briefly discuss the other variations of this formalization to accommodate different structures like trees and structured latent variables.

2.2.1 Syntactic Parsing

For parsing problems, the input is still a sequence of words, but the output is a tree. The output tree structure introduces several issues.

The first and most obvious problem is the much larger search space. There are $O(m^m)$ possible trees for a given input sequence of length m, which means exhaustive search is not possible in practice even for very short sentences. Several algorithms have been used to speedup the searching for the best parse derivations in practice.

1. For constituent parsing, tabular parsing, i.e., CKY parsing, is the default parsing algorithm. CKY parsing can be viewed as a 2-dimensional variation of previous Viterbi algorithm, where the candidates for each span in the sentence is kept in a table. Candidates for a new span is generated after all the spans that the new span depends on have been explored. For binarized grammar, CKY algorithm takes $O(m^3)$ time to parse a sentence. However, even with CKY algorithm, the $O(m^3)$ time complexity is still not affordable for long sentences and large dataset.

step	action	stack	buffer	arc added
0	-	root	The cat ...	
1	SHIFT	root The	cat sat ...	
2	SHIFT	root The cat	sat on ...	
3	LEFT-REDUCE	root cat	sat on ...	The$\curvearrowleft$cat
4	SHIFT	root cat sat	on the ...	
5a	LEFT-REDUCE	root sat	on the ...	cat$\curvearrowleft$sat
6a	SHIFT	root sat on	the mat	
7a	RIGHT-REDUCE	root sat	the mat	sat$\curvearrowright$on
8a	SHIFT	root sat the	mat	
9a	SHIFT	root sat the mat		
10a	LEFT-REDUCE	root sat mat		the$\curvearrowleft$mat
11a	RIGHT-REDUCE	root sat		sat$\curvearrowright$mat
5b	SHIFT	root cat sat on	the mat	
6b	RIGHT-REDUCE	root sat	the mat	sat$\curvearrowright$on
7b	SHIFT	root cat sat the	mat	
8b	SHIFT	root cat sat the mat		
9b	LEFT-REDUCE	root cat sat mat		the$\curvearrowleft$mat
10b	RIGHT-REDUCE	root cat sat		sat$\curvearrowright$mat
11b	LEFT-REDUCE	root sat		cat$\curvearrowleft$sat
12	RIGHT-REDUCE	root		root$\curvearrowright$sat

Table 2.1: Incremental dependency parsing transitions for sentence "The cat sat on the mat" to generate the dependency tree in Figure 1.2 (b). Two derivations that both reach the correct dependency parse are shown in this table. Their different transitions are marked with "a" and "b" respectively.

In practice, beam search is widely used to reduce the searching time.

2. Another widely used inference algorithm for both constituent parsing and dependency parsing is incremental parsing like shift-reduce parsing [39]. Take dependency parsing for an example. The shift-reduce parsing consists of a sequence of parsing states. Each parsing state is composed of a stack, a buffer, and an arc set. Parsing transitions are applied to parsing states to generate new states. There are three kinds of transitions: SHIFT, LEFT-REDUCE, and RIGHT-REDUCE. The SHIFT transition pops the top word from the buffer and pushes it onto the stack. The LEFT-REDUCE transition pops the top two items, s_0 and s_1, from the stack, adds an dependency relation $s_1 \curvearrowleft s_0$ into the arc set, and pushes s_0 onto the stack. The RIGHT-REDUCE transition is similar to LEFT-REDUCE, but adds arc $s_1 \curvearrowright s_0$ to the arc set instead. Exemplary parsing sequences for sentence "The cat sat on the mat" to generate the dependency tree in Figure 1.2 (b) are shown in Table 2.1. Shift-reduce parsing is usually combined with beam search to achieve linear inference time.

3. Some other inference algorithms are also available. One example is the Minimum Spanning Tree algorithm [37] that first assigns a penalty to every potential edges between each pair of words, and then finds a minimum spanning tree as the dependency tree. Linear programming is also used to find the tree that have the highest score, but under the tree structure constraint [36].

Another issue in the parsing problem is that, there can be more than one paths that generate the correct output tree, which is known as the *spurious ambiguity*. For example, in the example in Table 2.1, both of the two derivations can generate the correct parse tree. They differ in when to add the left arc pointing from "sat" to "cat", but it does not affect the final output tree. This kind of spurious ambiguity does *not* occur in the simple sequence labeling problem like POS tagging, since in the sequence labeling problem all of the searching steps are observed given the label sequence. But for the parsing problem, only the final output tree is annotated, rather than the complete search steps. Recall our discussion in Chapter 1.3, this is exactly another example of latent factors in our inference model. However, due to the simplicity of parsing derivations, we usually just ignore this spurious ambiguity problem and simply choose one fixed correct derivation

as the reference derivation as the inference goal.

2.2.2 Semantic Parsing & Machine Translation

Due to the similarities between the semantic parsing problem and the machine translation problem, we discuss them together. As we mentioned in Chapter 1.3, both of these two problems involve two parts of hidden factors: the rules to be applied and how these rules are composed. To directly annotate these two factors is impractical, and a more natural way is to annotate the output structure, i.e., lambda expressions or translated sentences. In other words, the spurious ambiguity problem in both semantic parsing and machine translation is more serious than in syntactic parsing problem, and to fully model this spurious ambiguity problem, latent models are introduced, which we will discuss in Chapter 4.

Besides the spurious ambiguity problem, the inference is also harder, especially for machine translation problem, due to the larger search space. In general, there are two paradigms of inference algorithms for machine translation. The first one is phrase-based machine translation [26], which generates the translated sentence in a left-to-right order. At each step, it picks a phrase in the source sentence, translates it and appends the translated phrase to the end of the partial translated target sentence. The other paradigm is syntax-based machine translation [59, 12]. Syntax-based machine translation is based on a synchronous grammar which is learned from the training corpus. At inference time, the source sentence is parsed following the left side of the synchronous grammar, while on the target side a translated sentence is generated following the right side of the synchronous grammar. Both of these two inference algorithms will be discussed in later chapters.

2.2.3 Sentence Entailment

Sentence entailment task differs from previous tasks in that the output of sentence entailment problem is *not* structured. However, the inference of the output label for sentence entailment problem involves a search in the structured space for tree node alignment and node-wise entailment labeling.

Example of the inference algorithm for sentence entailment are based on tree edits

[34, 57], where the premise tree is changed step by step to match the hypothesis tree, and at each step the edit can change the entailment relation between the two sentences.

Chapter 3: Structured Learning Algorithms

In this chapter we inspect the widely used structured learning algorithms, Structured perceptron [13], and Conditional Random Field (CRF) [31].

As we mentioned in Chapter 1.2, all structured learning algorithms have their original non-structured versions in the machine learning domain. We listed these algorithms with their non-structured versions in Table 3.1.

3.1 Structured Perceptron

The first structured learning algorithm we investigate is Structured Prediction, due to its extreme simplicity. However, in next chapter we will show that, even with such a simple learning method, some properties of the learning process is still unknown.

3.1.1 Binary Perceptron

We start from the standard perceptron for binary classification, and later extend it to structured classification.

Perceptron [46] is a binary classifier based on a linear prediction function:

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} > b \\ 0 & \text{otherwise} \end{cases},$$

where the input $\mathbf{x}$ is a vector, the parameter $\mathbf{w}$ is a weight vector, and b is bias. A widely used technique to simplify the representation is to augment $\mathbf{x}$ by appending a value 1 at the end:

$$\mathbf{x} = (x_1, x_2, \ldots, x_m, 1),$$

	binary/multi-class classification	structured learning
generative model	nave bayes	HMM
discriminative model	logistic regression (maximum entropy)	CRF
	perceptron	Structured perceptron
	SVM	structured SVM / M^3N

Table 3.1: Relations between various structured learning models and their non-structured versions. The last two columns illustrate the non-structured classifiers and how they are extend to their structured versions. For example, Hidden Markov Model (HMM) is a structured extension of the nave bayes model for binary classification.

Algorithm 1 Learning algorithm for perceptron.

Input: training set $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)}\} \in \mathcal{X} \times \mathcal{Y}$, number of iterations I
Output: weight vector $\mathbf{w}$
$\mathbf{w} \leftarrow (0, 0, \ldots, 0)$
for $i = 1, \ldots, I$ **do**
    for $j = 1, \ldots, N$ **do**
        if $y^{(j)}\mathbf{x}^{(j)} \cdot \mathbf{w} \leq 0$ **then**
            $\mathbf{w} \leftarrow \mathbf{w} + y^{(j)}\mathbf{x}^{(j)}$
return w

so that bias b can be dropped. The simplified prediction function is:

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Algorithm 1 shows the learning process of perceptron. In general, perceptron optimizes the parameter $\mathbf{w}$ to minimize the errors among the training examples in an online fashion. For each example, it checks whether current weight $\mathbf{w}$ can successfully classify it. If not, perceptron will update the parameter $\mathbf{w}$ by moving it towards the direction of the input $\mathbf{x}$.

A great property of perceptron is its convergence guarantee. It can be proved that for a linear separable dataset, perceptron will eventually find the optimal $\mathbf{w}$ that correctly classifies all examples in the dataset [46]. We show the proof here since it will be used later for structured cases.

Denote R to be the maximal radius of the training examples, $R = \max_i \|\mathbf{x}^{(i)}\|$. As-

sume the training set D is separable by a unit oracle weight vector $\mathbf{u}$ with margin γ:

$$\gamma = \max_{\mathbf{u}:\|\mathbf{u}\|=1} \min_{(\mathbf{x},y)\in D} y\mathbf{x}\cdot\mathbf{u}.$$

Theorem 1 (convergence of binary perceptron [46])**.** *For a separable dataset* $D = \{\mathbf{x}^{(i)}, y^{(i)}\}_i^N$ *with optimal margin* γ *and radius* R*, perceptron terminates after* t *updates, where* $t \leq \frac{R^2}{\gamma^2}$*.*

Proof. Let $\mathbf{w}^t$ be the weight vector **before** the t^{th} update. Suppose t^{th} update occurs on example $(\mathbf{x}^t, y^t)$. Following properties hold:

1. $y^t(\mathbf{u}\cdot\mathbf{x}^t) \geq \gamma$ (margin on unit vector)
2. $y^t(\mathbf{w}^t\cdot\mathbf{x}^t) \leq 0$ (Algorithm 1 Line 6)
3. $\|\mathbf{x}^t\|^2 \leq R^2$ (radius)

The update formula can be rewritten as:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + y^t\mathbf{x}^t. \tag{3.1}$$

$\|\mathbf{w}^{t+1}\|$ is bounded in two directions:

1. Dot product both sides of Equation 3.1 with oracle vector $\mathbf{u}$:

$$\begin{aligned}\mathbf{u}\cdot\mathbf{w}^{t+1} &= \mathbf{u}\cdot\mathbf{w}^t + y^t(\mathbf{u}\cdot\mathbf{x}^t)\\ &\geq \mathbf{u}\cdot\mathbf{w}^t + \gamma && \text{(margin)}\\ &\geq \sum_t \gamma = t\gamma && \text{(by induction)}\end{aligned}$$

Algorithm 2 Learning algorithm for averaged perceptron.

Input: training set $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)}\} \in \mathcal{X} \times \mathcal{Y}$, number of iterations I
Output: weight vector $\mathbf{w}$
$\mathbf{w} \leftarrow (0, 0, \ldots, 0)$
$\mathbf{w}_c \leftarrow (0, 0, \ldots, 0)$
$c \leftarrow 1$
for $i = 1, \ldots, I$ **do**
    for $j = 1, \ldots, N$ **do**
        if $y^{(j)}\mathbf{x}^{(j)} \cdot \mathbf{w} \leq 0$ **then**
            $\mathbf{w} \leftarrow \mathbf{w} + y^{(j)}\mathbf{x}^{(j)}$
            $\mathbf{w}_c \leftarrow \mathbf{w}_c + cy^{(j)}\mathbf{x}^{(j)}$
        $c \leftarrow c + 1$
return $\mathbf{w} - \mathbf{w}_c/c$

2. Take the norm of both sides of Equation 3.1:

$$\begin{aligned}
\|\mathbf{w}^{t+1}\|^2 =& \|\mathbf{w}^t + y^t\mathbf{x}^t\|^2 \\
=& \|\mathbf{w}^t\|^2 + \|y^t\mathbf{x}^t\|^2 + 2y^t\mathbf{w}^t \cdot \mathbf{x}^t \\
\leq& \|\mathbf{w}^t\|^2 + \|\mathbf{x}^t\|^2 && \text{(Property 2)} \\
\leq& \|\mathbf{w}^t\|^2 + R^2 && \text{(radius)} \\
=& \sum_t R^2 = tR^2 && \text{(by induction)}
\end{aligned}$$

Combining the two bounds for $\mathbf{w}^{t+1}$ leads to:

$$t^2\gamma^2 \leq \|\mathbf{w}^{t+1}\|^2 \leq tR^2,$$

which indicates $t \leq \frac{R^2}{\gamma^2}$. □

Theorem 1 gives a guarantee that perceptron will correctly classify the training set after a bounded number of updates. But the weight vector $\mathbf{w}$ learned by perceptron might be overfitted so that it can only classify the training set, while can not achieve similar performance when applied to other similar datasets. This problem is usually handled by using a separate development dataset to stop the training when perceptron overfits,

and/or adding regularizers to limit the complexity of the model. Average perceptron [20] can also ease this problem by averaging the weights learned at each step to achieve a *large margin effect* similar to SVM. Algorithm 2 shows the average perceptron algorithm which is adapted from [16].

3.1.2 Structured Extension

perceptron can be extended to handle structured learning problem [13]. Due to its simplicity, structured perceptron is a very popular structured learning algorithm.

Recall our formalization of the structured prediction problem in Chapter 2.1, where the input is a structure $x \in \mathcal{X}$ and the output is also a structure $y \in \mathcal{Y}$. For structured perceptron, the scoring function $F_{\mathbf{w}}$ is defined as:

$$F_{\mathbf{w}}(x, y) = \mathbf{\Phi}(x, y) \cdot \mathbf{w},$$

where $\mathbf{\Phi}$ is a feature function that extracts the structured features from structures x and y and maps them to a vector. In most cases, $\mathbf{\Phi}$ is defined to satisfy decomposability so that dynamic programming can be used in the inference problem.

One example feature function $\mathbf{\Phi}$ for the POS tagging task can be defined as:

$$\mathbf{\Phi}(x, y) = \sum_{i=1}^{m} \mathbf{\Phi}(x, y_{i-2}y_{i-1}y_i),$$

where we abuse the notation to use $\mathbf{\Phi}$ to represent the stepwise feature function over a partial label sequence. The stepwise scoring function $f_{\mathbf{w}}$ can be written as:

$$f_{\mathbf{w}}(y_i; x, y_{i-2}y_{i-1}) = \mathbf{\Phi}(x, y_{i-2}y_{i-1}y_i) \cdot \mathbf{w}.$$

An exemplary stepwise feature function can use the features defined in Table 3.2.

Under this definition, the objective function of the structured prediction is:

$$y^* = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}}\, \mathbf{\Phi}(x, y) \cdot \mathbf{w}.$$

The learning algorithm for structured perceptron is shown in Algorithm 3.

atomic features	x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i-2}, y_{i-2}, y_{i-1}, y_i
combined features	$y_{i-2}\|y_{i-1}\|y_i$, $y_{i-1}\|y_i$, $x_{i-2}\|y_i$, $x_{i-1}\|y_i$, $x_i\|y_i$, $x_{i+1}\|y_i$, $x_{i+2}\|y_i$, $x_{i-2}\|y_i$, $x_{i-2}\|y_{i-1}\|y_i$, $x_{i-1}\|y_{i-1}\|y_i$, $x_i\|y_{i-1}\|y_i$, $x_{i+1}\|y_{i-1}\|y_i$, $x_{i+2}\|y_{i-1}\|y_i$, $x_{i-2}\|y_{i-1}\|y_i$

Table 3.2: An exemplary feature set for POS tagging task.

Algorithm 3 Learning algorithm for structured perceptron.

Input: training set $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)}\}$, feature map $\mathbf{\Phi}$
Output: weight vector $\mathbf{w}$
Let $\Delta\mathbf{\Phi}(x, y, z) \stackrel{\Delta}{=} \mathbf{\Phi}(x, y) - \mathbf{\Phi}(x, z)$
$\mathbf{w} \leftarrow (0, 0, \ldots, 0)$
repeat
    for each $(x, y) \in D$ **do**
        $z \leftarrow \text{argmax}_{y' \in \mathcal{Y}(x)} \mathbf{\Phi}(x, y') \cdot \mathbf{w}$
        if $z \neq y$ **then**
            $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{\Phi}(x, y, z)$
until Converged
return w

3.1.3 Convergence of Structured Perceptron

In general, structured perceptron shares the same convergence property as binary perceptron. However, in practice approximated search is usually used by default in the decoding for structured perceptron, due to the exponentially large search space, which violates the conditions in Theorem 1. Figure 3.1 shows a counter example in POS tagging combined with beam search where the convergence guarantee is not satisfied.

This problem is first observed and handled with a strategy called "early update" in [14]. This strategy is based on the intuition that, in the beam search, the reference derivation should always survive the beam. So they update the weight vector $\mathbf{w}$ when the reference derivation falls out of the beam by penalizing the top derivation in the beam and rewarding the reference derivation. This strategy works well and is used in most structured perceptron applications by default. The theoretical explanation for this method is proposed in [23].

Dataset $D = \{(x, y)\}$

x	fruit	flies	fly	.
y	N	N	V	.

Search sapce $\mathcal{Y}(x) = \{\text{N}\} \times \{\text{N, V}\} \times \{\text{N, V}\} \times \{.\}$

Features $\mathbf{\Phi}(x, y) = (\#_{N \to N}(y), \#_{V \to .}(y))$

Training process:

iter	label z	$\Delta\mathbf{\Phi}(x, y, z)$	$\Delta\mathbf{\Phi} \cdot \mathbf{w}$	new $\mathbf{w}$
0				$(0, 0)$
1	N N N .	$(-1, +1)$	0	$(-1, 1)$
2	N V N .	$(+1, +1)$	0	$(0, 2)$
3	N N N .	$(-1, +1)$	2	$(-1, 3)$
4	N V N .	$(+1, +1)$	2	$(0, 4)$

... infinite loop ...

Figure 3.1: A POS tagging example where standard structured perceptron does not converge. The dataset only contains one sentence "fruit flies fly.". The simplified tagging set $\mathcal{T} = \{\text{N}, \text{V}, .\}$. The simplified feature function only counts the number of "N-N" tag bigrams ($\#_{N \to N}(y)$), and the number of "V-." tag bigrams ($\#_{V \to .}(y)$). The Viterbi decoding algorithm is greedy, i.e., beam size $b = 1$. [23]

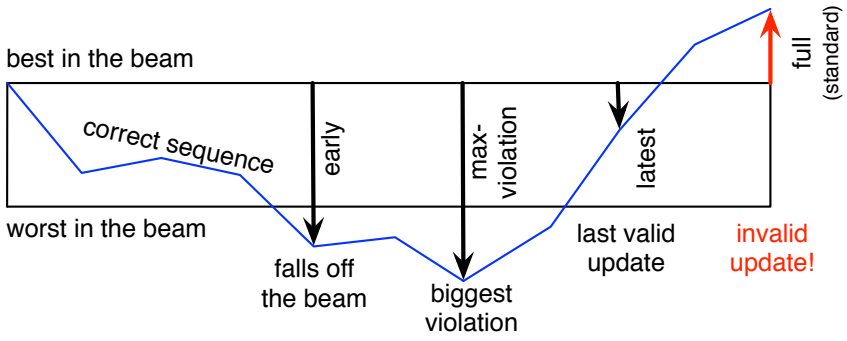


Figure 3.2: Update methods for structured perceptron with beam search.

To theoretically explain why the convergence guarantee does not hold, following [23], we rewrite the structured perceptron version of the convergence proof here.

First, in the step-by-step search setting, the training set D actually consists of prefixes of sequences, since the update can occur at any prefix of a given sequence.

Assume the training set D is separable by feature map $\mathbf{\Phi}$ and unit oracle vector $\mathbf{u}$ with optimal margin

$$\gamma = \max_{\mathbf{u}:\|\mathbf{u}\|=1} \min_{(x,y)\in D, z\neq y} \mathbf{u} \cdot \Delta\mathbf{\Phi}(x, y, z). \tag{3.2}$$

Let $R = \max_{(x,y)\in D, z\neq y} \Delta\mathbf{\Phi}(\mathbf{x}, y, z)$ be the radius.

Theorem 2 (convergence of structured perceptron [23])**.** *For a separable dataset $D = \{x^{(i)}, y^{(i)}\}_i^N$ with optimal margin γ and radius R under feature map $\mathbf{\Phi}$, structured perceptron terminates after t updates, where $t \leq \frac{R^2}{\gamma^2}$.*

Proof. Let $\mathbf{w}^t$ be the weight vector **before** the t^{th} update. Suppose t^{th} update occurs on example (x^t, y^t), and the top candidate $z \neq y^t$.

Following properties hold:

1. $\mathbf{u} \cdot \Delta\mathbf{\Phi}(x^t, y^t, z) \geq \gamma$ (margin on unit vector)
2. $\mathbf{w}^t \cdot \Delta\mathbf{\Phi}(x^t, y^t, z) < 0$ (Algorithm 3 Line 7)
3. $\|\Delta\mathbf{\Phi}(x^t, y^t, z)\|^2 \leq R^2$ (radius)

The update formula is:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \Delta\mathbf{\Phi}(x^t, y^t, z). \tag{3.3}$$

$\|\mathbf{w}^{t+1}\|$ is bounded in two directions.

1. Dot product both sides of Equation 3.3 with oracle vector $\mathbf{u}$:

$$\begin{aligned} \mathbf{u} \cdot \mathbf{w}^{t+1} &= \mathbf{u} \cdot \mathbf{w}^t + \mathbf{u} \cdot \Delta\mathbf{\Phi}(x^t, y^t, z) \\ &\geq \mathbf{u} \cdot \mathbf{w}^t + \gamma && \text{(margin)} \\ &\geq \sum_t \gamma = t\gamma && \text{(by induction)} \end{aligned}$$

2. Take the norm of both sides of Equation 3.3:

$$\begin{aligned}
\|\mathbf{w}^{t+1}\|^2 =& \|\mathbf{w}^t + \Delta\mathbf{\Phi}(x^t, y^t, z)\|^2 \\
=& \|\mathbf{w}^t\|^2 + \|\Delta\mathbf{\Phi}(x^t, y^t, z)\|^2 + 2\mathbf{w}^t \cdot \Delta\mathbf{\Phi}(x^t, y^t, z) \\
\leq& \|\mathbf{w}^t\|^2 + \|\Delta\mathbf{\Phi}(x^t, y^t, z)\|^2 && \text{(Property 2)} \\
\leq& \|\mathbf{w}^t\|^2 + R^2 && \text{(radius)} \\
=& \sum_t R^2 = tR^2 && \text{(by induction)}
\end{aligned}$$

Combining the two bounds for $\mathbf{w}^{t+1}$ leads to:

$$t^2\gamma^2 \leq \|\mathbf{w}^{t+1}\|^2 \leq tR^2,$$

which indicates $t \leq \frac{R^2}{\gamma^2}$. □

To theoretically explain why standard perceptron does not always converge with beam search, note that Property 2 in the above proof, which is called "violation" in [23], is required in the second part of the proof. However, this is not necessarily always true. As shown in Figure 3.1, $\Delta\mathbf{\Phi} \cdot \mathbf{w}$ is positive for some updates, which violates the convergence property.

Furthermore, this observation points out that, as long as the update satisfies the violation property, the convergence property holds.

For early update, since at the step where the reference falls out of the beam, the reference candidate y ranks lower than the top candidate z:

$$\mathbf{\Phi}(x, y) \cdot \mathbf{w} \leq \mathbf{\Phi}(x, z) \cdot \mathbf{w} \Rightarrow \Delta\mathbf{\Phi}(x, y, z) \cdot \mathbf{w} \leq 0.$$

Thus, the convergence is still guaranteed.

This observation further points out a new update strategy, "max-violation" update [23], which updates at the step where the score difference between the partial reference candidate and the top candidate in the beam is maximal (Figure 3.2). The intuition behind this is that at such step the structured perceptron can learn most, and, thus, converges faster. However, there is no theoretical proof for the convergence of max-violation update.

3.2 Conditional Random Field

Conditional Random Field (CRF) [31] is another popular discriminative learning algorithm for structured prediction problems. It is designed to handle general structured prediction problem. But here we only discuss a simple version of CRF for sequence labeling problem.

3.2.1 Conditional Random Field with Exact Search

As a discriminative model, CRF models the probability of an output $y \in \mathcal{Y}(x)$ as:

$$\Pr(y|x) = \frac{1}{Z(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, y)],$$

where the denominator $Z(x)$ normalizes the result to a probability distribution:

$$Z(x) = \sum_{y' \in \mathcal{Y}(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, y')],$$

and $\mathbf{\Phi}$ is a feature function defined similarly to the feature function in perceptron.

The scoring function $F_{\mathbf{w}}(x, y)$ for CRF now is defined to be the probability of output y given input x: $F_{\mathbf{w}}(x, y) = \Pr(y|x)$. For the inference problem, the denominator $Z(x)$ is usually ignored since it is shared by all possible output y. So the inference problem is simplified to find the output y with maximum $\mathbf{w} \cdot \mathbf{\Phi}(x, y)$. In this case, as long as the feature function $\mathbf{\Phi}$ is decomposable to be the summation over features of partial sequence labels, the Viterbi inference algorithm is still applicable to find the optimal output sequence.

For the learning problem, the formal objective function for CRF learning is to minimize the *negative conditional log likelihood* over the training examples:

$$\begin{aligned} \ell(\mathbf{w}) &= -\sum_{(x,y) \in D} \log \Pr(y|x) \\ &= -\sum_{(x,y) \in D} \log \frac{1}{Z(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, y)] \end{aligned} \tag{3.4}$$

which is a convex function and can be optimized by methods like improved iterative

Algorithm 4 SGD Learning algorithm for CRF.

Input: training set $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)}\}$, feature map $\mathbf{\Phi}$
Output: weight vector $\mathbf{w}$
$\mathbf{w} \leftarrow (0, 0, \ldots, 0)$
repeat
  for each $(x, y) \in D$ **do**
    Calculate $\Pr(y'|x) \leftarrow \frac{1}{Z(x)} \exp[\mathbf{\Phi}(x, y') \cdot \mathbf{w}]$ for $\forall y' \in \mathcal{Y}(x)$
    $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{\Phi}(x, y) - \sum_{y' \in \mathcal{Y}(x)} \Pr(y')\mathbf{\Phi}(x, y')$
until Converged
return w

scaling [6], gradient descent, and quasi-newton methods.

Here we derive the gradient for the objective function in Equation 3.4:

$$\begin{aligned}\frac{\partial \ell(\mathbf{w})}{\partial \mathbf{w}} &= \frac{-\sum_{(x,y\in D)} \partial \log \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, y)] - \partial \log Z(x)}{\partial \mathbf{w}} \\ &= -\sum_{(x,y)\in D} [\mathbf{\Phi}(x, y) - \sum_{y' \in \mathcal{Y}(x)} \Pr(y'|x)\mathbf{\Phi}(x, y')]. \end{aligned} \tag{3.5}$$

CRF is usually trained in a batch mode with L-BFGS, a quasi-newton method. Here, to stress its similarity between perceptron, we instead choose stochastic gradient descent (SGD) training, which is also widely used [48, 18, 70]. The key idea is that with SGD there can be a lot more updates on the model parameters so ideally the model can be trained faster.

The stochastic objective function for CRF is

$$\ell(\mathbf{w}; x, y) = -\log \Pr(y|x) = -\log \frac{1}{Z(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, y)]$$

The stochastic gradient (corresponding to Equation 3.5) is

$$\frac{\partial \ell(\mathbf{w}; x, y)}{\partial \mathbf{w}} = -\mathbf{\Phi}(x, y) + \sum_{y' \in \mathcal{Y}(x)} \Pr(y'|x)\mathbf{\Phi}(x, y') \tag{3.6}$$

The SGD version of CRF is shown in Algorithm 4.

Both batch training and stochastic training for standard CRF are guaranteed to

converge since the objective function of CRF (Equation 3.4) is convex.

3.2.2 Conditional Random Field with Inexact Search

The major issue remains with stochastic gradient descent is about the second term in Equation 3.6 (Line 6 in Algorithm 4). The update step needs to iterative over all possible output structures in $\mathcal{Y}(x)$, calculates the probability of each structure, and combines the structure features weighted by the corresponding probabilities. Calculating this probability can be done with dynamic programming algorithm similar to the Viterbi Algorithm, but still the time needed is not affordable, since there can be exponentially many output structures.

A straightforward alternative to solve this issue is beam search. Many works using CRF have adapted beam search with stochastic update [48, 18, 70]. With beam search, in the searching for the possible output structures $y' \in \mathcal{Y}(x)$, only the top b structures are kept and used later to calculate the probability distribution $\Pr(y'|x)$ under current model weight parameter. More formally we can define the new output space under beam search:

$$\tilde{\mathcal{Y}}_{\mathbf{w}}(x) = \mathbf{B}_m(x; \mathbf{w}),$$

where $\mathbf{B}_m(x; \mathbf{w})$ is the set of candidates that survives the beam search defined in Equation 2.2.

Despite the wide usage and good performance of CRF with beam search, there was relatively little known about its theoretical properties, especially about the convergence.

A recent research [48] discuss about the convergence property of CRF with beam search. Here we just briefly state the results here[1] and skip the proof. We will come back to the proof again in later chapter.

Denote $\mathbf{w}^*$ to be the optimal weight vector that minimizes the conditional log likelihood objective of CRF with *exact* search:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \ell(\mathbf{w}). \tag{3.7}$$

It is shown in [48] that $\ell(\mathbf{w}^*)$ can be approximated by CRF with beam search with

[1] Our formalization is slightly different from the formalization in [48] since we do not have the regularization term in the objective function and we do not consider the learning rate.

arbitrary precision as long as the beam size b is sufficiently large.

With exact search, the stepwise SGD update term is shown in Equation 3.6. We rewrite it here:

$$\nabla \ell(\mathbf{w}; x, y) = \frac{\partial \ell(\mathbf{w}; x, y)}{\partial \mathbf{w}} = -\mathbf{\Phi}(x, y) + \sum_{y' \in \mathcal{Y}(x)} \Pr(y'|x) \mathbf{\Phi}(x, y')$$

With beam search, the stepwise SGD update term is:

$$\nabla \tilde{\ell}(\mathbf{w}; x, y) = -\mathbf{\Phi}(x, y) + \sum_{y' \in \tilde{\mathcal{Y}}_{\mathbf{w}}(x)} \Pr(y'|x) \mathbf{\Phi}(x, y').$$

The difference between the exact update term and inexact update term

$$\delta(\mathbf{w}; x, y) = \nabla \ell(\mathbf{w}; x, y) - \nabla \tilde{\ell}(\mathbf{w}; x, y),$$

can be arbitrarily close to 0 with a sufficiently large beam size b:

$$\delta(\mathbf{w}; x, y) \leq \epsilon.$$

Let τ be an approximation-based bound of the difference between $\ell(\mathbf{w})$ and $\tilde{\ell}(\mathbf{w})$ s.t.

$$[\nabla \ell(\mathbf{w}) - \nabla \tilde{\ell}(\mathbf{w})] \cdot (\mathbf{w} - \mathbf{w}^*) \leq \tau,$$

where $\nabla \tilde{\ell}(\mathbf{w}) \triangleq \mathbb{E}_{(x,y)}[\nabla \tilde{\ell}(\mathbf{w}; x, y)]$, which is the expected update term over all data for a given weight $\mathbf{w}$. τ is controlled by the beam size b since increasing b can make $\nabla \tilde{\ell}(\mathbf{w})$ arbitrary-close to $\nabla \ell(\mathbf{w})$.

We also need some assumptions about the properties of the objective function:

1. $\ell(\mathbf{w})$ is strongly convex with modulus c,: $\forall \mathbf{w}, \mathbf{w}'$,

$$\ell(\mathbf{w}') \geq \ell(\mathbf{w}) + (\mathbf{w}' - \mathbf{w}) \nabla \ell(\mathbf{w}) + \frac{c}{2} \|\mathbf{w}' - \mathbf{w}\|^2.$$

2. $\nabla \ell(\mathbf{w})$ is Lipschitz continuous differentiable with constant q: $\forall \mathbf{w}, \mathbf{w}'$,

$$\|\nabla \ell(\mathbf{w}') - \nabla \ell(\mathbf{w})\| \leq q \|\mathbf{w}' - \mathbf{w}\|.$$

Then we have the convergence theorem for CRF with beam search:

Theorem 3 (convergence of CRF with beam search [48])**.** *Let ϵ be a target degree of convergence, CRF with beam search will converge if beam size b satisfies*

$$\tau \leq \frac{c\epsilon}{2q}.$$

More precisely, let $\mathbf{w}^0$ be the initial weight vector, after t updates where

$$t \geq \frac{\log(\frac{q\|\mathbf{w}^0 - \mathbf{w}^*\|^2}{\epsilon})}{c},$$

CRF with beam search converges towards the optimum objective s.t.

$$\mathbb{E}[\ell(\mathbf{w}^t) - \ell(\mathbf{w}^*)] \leq \epsilon.$$

This theorem gives the convergence guarantee for CRF with beam search where only the top b candidate output structures are sampled. It states that, as long as the beam size b is sufficiently large, CRF with beam search will converge.

3.2.3 Conditional Random Field vs. Structured Perceptron

The differences between the update functions of structured perceptron (Equation 3.3) and CRF optimized with SGD (Equation 3.6) are relatively small: they both reward the features from the correct output derivation, and penalize the incorrect derivation(s). Actually we can approximate structured perceptron by CRF with SGD by using the Viterbi output as an approximation of the set of all possible output structures weighted by the probabilities. More specially, structured perceptron can be viewed as CRF with SGD and beam search of beam size $b = 1$.

To summarize, structured perceptron can be viewed as a two-fold approximation of standard CRF: first an online approximation using SGD, and second a hard Viterbi approximation (using beam search with beam size 1).

Chapter 4: Latent Variable Structured Learning: Theory

As we mentioned in Chapter 1.3, for many structured learning tasks, only part of the output derivation is easy to annotate by human. With only partially known reference, the structured learning problem is more difficult, but also becomes capable of dealing with more complicated tasks.

The general approach for structured learning with natural annotations is to model the unknown full annotations as latent variables, and search for the correct full annotations that agree with the partially known references. This latent variables are structured just like the structured output, which means even bigger search space, further slowing down the decoding process. More importantly, the structured latent variables in the search space violates many theoretical properties for the structured learning algorithms in Chapter 3, so that the theoretical convergence properties no longer always hold for structured perceptron and CRF with latent variables. Furthermore, we investigate the application of inexact search in the expectation-maximization algorithm in NLP, which also involves structured latent variables, but is generative instead of discriminative like perceptron and CRF.

4.1 Mathematical Formalization for Latent Variable Sequence Labeling

We continue our formalization for the simple sequence labeling problem in Chapter 2.1 and augment it with a latent variable model. We use this example to establish the mathematical framework for latent variable structured learning, extend it to other structured learning tasks, and then discuss the theoretical properties of latent variable structured learning models.

The key difference between a sequence labeling model with and without latent variables is that, for latent variable models, the output of the decoding procedure is not the labeled output y, but a latent label $h \in \mathbf{H}(y)$ instead. $\mathbf{H}$ is a mapping function that maps a label y to the set of structured latent variables that are consistent with y.

Usually y can be induced from $h \in \mathbf{H}(y)$ easily. We will show two examples of this latent variable mapping later in this section.

With this latent variable mapping, the decoding procedure does not directly search for output label y, but for the latent representation h. We can redefine the inference/decoding problem in Chapter 2.1 in two ways:

1. We use the label sequence y corresponding to the latent label sequence h with the hight score as the output:

$$y^* = \underset{y \in \mathcal{Y}(x) \text{ s.t. } h \in \mathbf{H}(y)}{\operatorname{argmax}} F_{\mathbf{w}}(x, h). \tag{4.1}$$

2. We use the label sequence y whose corresponding latent label sequences have the highest aggregated score:

$$y^* = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}} \sum_{h \in \mathbf{H}(y)} F_{\mathbf{w}}(x, h). \tag{4.2}$$

For the correct output y, there can be more than one corresponding latent label sequences $h \in \mathbf{H}(y)$. All of these hs are considered *correct* since from them we can induce y equally. But still we can have a special inference problem which we call *constrained inference* or *forced decoding* which searches for the correct latent label sequence with the highest score that matches the output y:

$$h^* = \underset{h \in \mathbf{H}(y)}{\operatorname{argmax}} F_{\mathbf{w}}(x, h). \tag{4.3}$$

In Chapter 1.3, we present several exemplary structured learning tasks like machine translation, semantic parsing, and sentence entailment, for which introducing latent variables is straightforward and reasonable due to the necessity of natural annotations. For the very simple sequence labeling task like POS tagging, it seems there is no need for latent variables at first glance. However, actually we can augment the sequence labeling model with latent variables either for better modeling of the labeling procedure (Chapter 4.1.1), or for better generalization of the labeling procedure (Chapter 4.1.2).

Here we give two examples of the latent variable mapping, corresponding to two different variants of the sequence labeling problem.

4.1.1 Decomposable Latent Variable Sequence Labeling

Consider the fact that the POS tags represent the functionalities of the words in the sentence. It is quite reasonable that a more refined representation of these functionalities might model the sentence better. In other words, a more refined label set might model the functionalities with higher accuracy.[1] This is particularly helpful for sequence labeling applications like POS tagging over the the Chinese TreeBank [61] tag set where the granularity of POS tag set is not refined enough.

Following this intuition, we improve the sequence labeling model with an augmented label set that is hidden from the annotator. For example, we can split the POS tag NN for nouns in singular form to two different POS tags: NN_1 for countable nouns and NN_2 for uncountable nouns which can help determine the tag of the following verb. But in the perspective of the annotator, these two tags are the same and both countable nouns and uncountable nouns will be annotated as NN.

More formally, for a given label set $\mathcal{T}$, there is a mapping function $\mathbf{H}$ that maps a label $y_i \in \mathcal{T}$ to a set of k latent labels:

$$\mathbf{H}(y_i) = \{h_{y_i,1}, h_{y_i,2}, \ldots, h_{y_i,k}\}. \tag{4.4}$$

We assume the mapped latent label sets for different labels $y_i, y_j \in \mathcal{T}$ do not overlap

$$\mathbf{H}(y_i) \cap \mathbf{H}(y_j) = \emptyset. \tag{4.5}$$

For a sentence $x = (x_1 x_2 \ldots x_m)$ and annotated label sequence $y = (y_1 y_2 \ldots y_m)$, the corresponding latent label sequence is $h = (h_1 h_2 \ldots h_m)$ where $h_i \in \mathbf{H}(y_i)$.

We abuse the notation of $\mathbf{H}$ to define the mapping over a sequence of the original label set: $\mathbf{H}(y) = \mathbf{H}(y_1) \times \mathbf{H}(y_2) \times \cdots \times \mathbf{H}(y_m)$. We also use $\mathbf{H}(\mathcal{T})$ to denote the split label set from the reference label set $\mathcal{T}$.

This type of latent model splits the output labels into disjoint latent labels. The key property of it is that each output structure can be *decomposed* into sub-structures (e.g., prefixes in the POS tagging problem) that directly correspond to several latent sub-derivations. In other words, *the output structure casts strong and direct constraints*

[1] This observation also works for constituent parsing tasks where a refined label set can improve parsing accuracy [44].

on the structure of the derivations in the latent space.

In the decomposable latent variable structured learning, for each feature in the latent variable model with augmented latent labels, we can easily determine the corresponding feature with the unaugmented label set, because between the output derivation and the latent derivation, the only variance is the label set, and the structure remains the same. For example, for a feature containing only NN_1, its corresponding feature with the unaugmented label set can be derived by replacing NN_1 with the unsplit label NN. Based on this observation, it seems to be helpful to leverage the information in the (unaugmented) output space to help prove the convergence in the (augmented) latent space, which is the key intuition behind the proof of Theorem 4.

4.1.2 Non-Decomposable Latent Variable Sequence Labeling

A more difficult variant of the sequence labeling problem is the grapheme-to-phoneme conversion problem [7]. In grapheme-to-phoneme conversion, each grapheme (letter) is labeled with its phoneme (pronunciation). However, some graphemes can be silent, which means they do not have corresponding phonemes; some graphemes can have more than one corresponding phonemes, for example grapheme "x". Also the phonemes are annotated at word-level, i.e., the grapheme-phoneme alignment is not annotated.

Take the word "fixing" for example, its annotated pronunciation is [fɪksɪŋ]. However, the correspondences (alignments) between the graphemes and the phonemes are unknown. There can be several alignments for the pronunciation of "fixing". One example is:

"fixing"	=	f	i	x	ing
[fɪksɪŋ]		[f]	[ɪ]	[ks]	[ɪŋ]

Another example is:

"fixing"	=	f	i	x	-	i	n	g
[fɪksɪŋ]		[f]	[ɪ]	[k]	[s]	[ɪ]	-	[ŋ]

The goal of grapheme-to-phoneme conversion is to learn a model to correctly convert a word to its pronunciation.

We formalize the grapheme-to-phoneme conversion problem following [7]. Denote $x = (x_1, \ldots, x_m), x_i \in \mathcal{X}$ as the word, and $y = (y_1, \ldots, y_{m'}), y_i \in \mathcal{Y}$ as the annotated pronunciation. For a given word x, the inference or decoding problem is to find:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} F_{\mathbf{w}}(x, y),$$

where $\mathcal{Y}(x)$ is the set of possible output phonemes, and $F_{\mathbf{w}}$ is the scoring function.

We can view the grapheme-to-phoneme conversion problem as a co-segmentation problem, i.e., the grapheme and its corresponding phoneme are grouped together as a co-segment. There can be more than one schemes to co-segment the word and its pronunciation. For example, in the first example of word "fixing" there are 4 co-segments, while in the second example there are 7 co-segments. The co-segment scheme is latent and we denote it with $h = (h_1, \ldots, h_k) \in \mathbf{H}(x, y)$, where k is the number of co-segments, and each h_i is actually a pair of $(x_{i,1} \ldots x_{i,m_i}, y_{i,1} \ldots y_{i,m'_i})$ s.t. 1) the co-segments cover all graphemes and phonemes; and 2) different co-segments do not overlap.

The score of a joint word-pronunciation pair $\Pr(x, y)$ under a specific co-segment $h \in \mathbf{H}(x, y)$ can be modeled as a score based on all co-segments:

$$F_{\mathbf{w}}(h) = F_{\mathbf{w}}(h_1, \ldots, h_k). \tag{4.6}$$

The score of a joint word-pronunciation pair $\Pr(x, y)$ can be calculated by summing over all possible co-segment schemes:

$$F_{\mathbf{w}}(x, y) = \sum_{h \in \mathbf{H}(x,y)} F_{\mathbf{w}}(h).$$

Note that this summarization corresponds to the second definition of the decoding problem in latent variable structured learning (Equation 4.2).

Comparing with the decomposable latent variable learning, it is difficult to *decompose* the output structure to sub-structures that directly correspond to structures in the latent space. In other words, there are no *strong and direct* constraints from the output structure to restrict the searching in the latent space. For the grapheme-to-phoneme conversion task, the output structure, i.e., the sequence of phonemes, does not provide sufficient constraints on the latent structures, i.e., the sequence of phonemes and the

segmentation. Without this strong constraint, the convergence of the non-decomposable latent variable structured learning is a totally new problem.

4.2 Latent Variable Structured Perceptron

For structured perceptron, we usually choose the first version of the inference problem (Equation 4.1):

$$y^* = \underset{y \in \mathcal{Y}(x) \text{ s.t. } h \in \mathbf{H}(y)}{\operatorname{argmax}} F_{\mathbf{w}}(x, h).$$

Introducing the latent variable means the reference derivation is no longer fixed. In this case, we can choose to update the perceptron by rewarding the correct latent derivation with the highest score, and penalizing the incorrect Viterbi latent derivation.

The formal update function is:

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \Delta\mathbf{\Phi}(x, h, \tilde{h}), \tag{4.7}$$

where

$$h = \underset{h \in \mathbf{H}(y)}{\operatorname{argmax}} \mathbf{w}^t \cdot \mathbf{\Phi}(x, h),$$

$$\tilde{h} = \underset{h \in \vec{(}y'), y' \in \mathcal{Y}(x)}{\operatorname{argmax}} \mathbf{w}^t \cdot \mathbf{\Phi}(x, h).$$

For the convergence of latent variable structured perceptron, we start with our example of latent variable sequence labeling based on [49], and then we will try to generalize the conclusion to general latent variable structured prediction tasks.

It is proved that the convergence guarantee for decomposable latent variable structured perceptron in the sequence labeling task (Chapter 4.1.1) still holds [49]. More specially, this convergence proof guarantees that, if the latent variables model the latent label set $\mathbf{H}(\mathcal{T})$ which is split from label set $\mathcal{T}$, and the structured perceptron for the original problem with label set $\mathcal{T}$ converges, then the new latent variable structured perceptron also converges.

The key intuition behind the proof is that the new separation margin $\gamma_{\mathbf{H}}$ with latent label set $\mathbf{H}(\mathcal{T})$ can be induced from the original separation margin γ.

Recall the original separation margin γ is defined in Equation 3.2:

$$\gamma = \max_{\mathbf{u}:\|\mathbf{u}\|=1} \min_{(x,y)\in D, z\neq y} \mathbf{u} \cdot \Delta\mathbf{\Phi}(x, y, z).$$

The new separation margin with the latent labels is:

$$\gamma_{\mathbf{H}} = \max_{\mathbf{u}:\|\mathbf{u}\|=1} \min_{(x,y)\in D, h\in\mathbf{H}(y), h'\in\mathbf{H}(z), y\neq z} \mathbf{u} \cdot \Delta\mathbf{\Phi}(x, h, h').$$

Consider the feature mapping $\mathbf{\Phi}$ for the latent label set. This feature function $\mathbf{\Phi}$ is actually also "augmented" from the old feature space to a new feature space to accommodate the enlarged label set. More specially, a feature ϕ_i that uses the unigram label y_i is augmented to a new set of possible features using $h_i \in \mathbf{H}(y_i)$, i.e., the number of new augmented features is $k_i = |\mathbf{H}(y_i)|$. Similarly consider a feature ϕ_j that used label bigrams of y_j and y'_j. The label set augmentation introduces $k_j = |\mathbf{H}(y_j)| \cdot |\mathbf{H}(y'_j)|$ new features. For the original feature mapping $\mathbf{\Phi}$ of n features, in general feature ϕ_i is augmented to k_i features where $0 < i \leq n$.

Theorem 4 (Separation Margin of Sequence Labeling with Latent Splitting Label Set [49])**.** *Given the feature space augmentation vector* $\mathbf{k} = (k_1, k_2, \ldots, k_n)$*, if the original feature space can be separated with an oracle unit vector* $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ *by a separation margin* γ*, then the new latent feature space can be separated by a margin*

$$\gamma_{\mathbf{H}} \geq \gamma/T$$

where $T = \sqrt{\sum_{i=1}^{n} k_i u_i^2}$.

Proof. Construct a new vector $\mathbf{u_H}$ of length $\sum_{i=1}^{n} k_i$ from the oracle unit vector $\mathbf{u}$ in the original feature space as follows:

$$\mathbf{u_H} = (\overbrace{u_1, \ldots, u_1}^{k_1}, \overbrace{u_2, \ldots, u_2}^{k_2}, \ldots, \overbrace{u_n, \ldots, u_n}^{k_n}).$$

Consider latent feature vector $\mathbf{\Phi}(x, h) = (\beta_1^1, \ldots, \beta_1^{k_1}, \beta_2^1, \ldots, \beta_2^{k_2}, \ldots, \beta_n^1, \ldots, \beta_n^{k_n})$ of length $\sum_{i=1}^{n} k_i$ where $h \in \mathbf{H}(y)$. The feature vector in the original space is $\mathbf{\Phi}(x, y) = (\alpha^1, \alpha^2, \ldots, \alpha^n)$. Since the latent feature is mapping from the original feature, and each feature is an indicator of either 0 or 1, $\alpha_i = \sum_{j=1}^{k_i} \beta_i^j$.

Then for $h \in \mathbf{H}(y)$ and $h' \in \mathbf{H}(y')$

$$\begin{aligned}
& u_{\mathbf{H}} \cdot \Delta\mathbf{\Phi}(x, h, h') \\
=& \sum_{i=1}^{n} (u_i \sum_{j=1}^{k_i} (\beta_i^j - \beta'^j_i)) \\
=& \sum_{i=1}^{n} u_i(\alpha_i - \alpha'_i) \\
=& \mathbf{u} \cdot \Delta\mathbf{\Phi}(x, y, y') \geq \gamma
\end{aligned}$$

This means $\mathbf{u_H}$ is an oracle vector that separates every training example in the latent feature space by margin γ. Thus there exists an oracle unit vector $\mathbf{u_H}/T$ that separates every training example in the latent feature space by margin $\gamma_{\mathbf{H}} = \gamma/T$ where $T = \sqrt{\sum_{i=1}^{n} k_i u_i^2}$. □

With this new separation margin, the proof from Theorem 2 can be used to prove the convergence of structured perceptron for latent variable sequence labeling task.

4.3 Latent Variable CRF

As we discussed in Chapter 3.2.3, structured perceptron can be viewed as a two-step approximation of CRF. Applying CRF for sequence labeling is straightforward. Knowing that the convergence guarantee no longer holds for structured perceptron, we also analyze the impact of latent variable for CRF learning.

Note that there are two definitions of the inference problem for latent variable structured learning.

With the first definition (Equation 4.1), the inference problem of CRF is:

$$\begin{aligned}
y^* &= \underset{y \in \mathcal{Y}(x) \text{ s.t. } h \in \mathbf{H}(y)}{\operatorname{argmax}} \Pr(h|x) \\
&= \underset{y \in \mathcal{Y}(x) \text{ s.t. } h \in \mathbf{H}(y)}{\operatorname{argmax}} \frac{1}{Z(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, h)], \qquad (4.8)
\end{aligned}$$

where

$$Z(x) = \sum_{y' \in \mathcal{Y}(x)} \max_{h \in \mathbf{H}(y)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, h)].$$

The new negative conditional log likelihood objective function is:

$$\begin{aligned} \ell(\mathbf{w}) &= -\sum_{(x,y) \in D} \log \max_{h \in \mathbf{H}(y)} \Pr(h|x) \\ &= -\sum_{(x,y) \in D} \log \max_{h \in \mathbf{H}(y)} \frac{1}{Z(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, h)]. \end{aligned}$$

Clearly the introduction of the latent variable h makes the new objective function no longer convex. So the convergence proofs for original CRF and for CRF with inexact search no longer hold.

On the other hand, with the second definition of the inference problem (Equation 4.2), the inference problem of CRF is:

$$\begin{aligned} y^* &= \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \sum_{h \in \mathbf{H}(y)} \Pr(h|x) \\ &= \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \sum_{h \in \mathbf{H}(y)} \frac{1}{Z(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, h)], \end{aligned} \tag{4.9}$$

where

$$Z(x) = \sum_{y' \in \mathcal{Y}(x)} \sum_{h \in \mathbf{H}(y)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, h)].$$

The corresponding negative conditional log likelihood objective function is:

$$\begin{aligned} \ell(\mathbf{w}) &= -\sum_{(x,y) \in D} \log \sum_{h \in \mathbf{H}(y)} \Pr(h|x) \\ &= -\sum_{(x,y) \in D} \log \sum_{h \in \mathbf{H}(y)} \frac{1}{Z(x)} \exp[\mathbf{w} \cdot \mathbf{\Phi}(x, h)], \end{aligned}$$

which is still not a convex function.

At first glance under the second definition the CRF learning objective is preferable in the theoretical perspective since it is checks all latent states that are correct. However, in practice, enumerating all latent structures $h \in \mathbf{H}(y)$ is still time consuming and is

usually calculated with approximation like beam search. In this scenario, the inference problem under the first definition (Equation 4.8) is a special case of the inference problem under the second definition (Equation 4.9).

4.4 Expectation-Maximization with Inexact Search

Another widely used machine learning algorithm in NLP is the Expectation-Maximization (EM) algorithm. Different from structured perceptron and CRF, EM model by default includes latent variables, which makes its objective non-convex. In practice for efficiency considerations we usually apply inexact search for the latent variables in EM, which makes the convergence even harder to predict.

Here we first introduce some EM examples and then discuss the properties of EM with inexact search in the expectation step.

We formalize the sequence labeling problem following [7] with a generative model. Denote $x = (x_1, \ldots, x_m), x_i \in \mathcal{X}$ as the word, and $y = (y_1, \ldots, y_{m'}), y_i \in \mathcal{Y}$ as the annotated label. For the POS tagging task, $m' = m$, while for the grapheme-to-phoneme conversion task, $m' \neq m$. For a given word x, we aim to find:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \Pr(x, y),$$

where $\mathcal{Y}(x)$ is the set of possible output labels.

We denote the latent labels with $h = (h_1, \ldots, h_k) \in \mathbf{H}(x, y)$, where k is the number of latent labels. For the POS tagging task, each h_i is a latent tag, $k = m$. For the grapheme-to-phoneme conversion task, each h_i is actually a co-segment, i.e., a pair of $(x_{i,1} \ldots x_{i,m_i}, y_{i,1} \ldots y_{i,m'_i})$ s.t. 1) the co-segments cover all graphemes and phonemes; and 2) different co-segments do not overlap.

The probability of a sentence-label pair $\Pr(x, y)$ under a specific latent label $h \in \mathbf{H}(x, y)$ can be written as:

$$\Pr(h; x, y) = \Pr(h_1, \ldots, h_k). \tag{4.10}$$

The probability of a joint sentence-label pair $\Pr(x, y)$ can be calculated by summing over

all possible latent label schemes:

$$\Pr(x, y) = \sum_{h \in \mathbf{H}(x,y)} \Pr(h).$$

Note that this summarization corresponds to the second definition of the decoding problem in latent variable structured learning (Equation 4.2).

Similar to the sequence labeling problem, we can simplify the calculation of Equation 4.10 by applying a M-gram Markov assumption:

$$\Pr(h; x, y) = \prod_i \Pr(h_i | h_{i-M+1}, \ldots, h_{i-1}),$$

where $\Pr(h_i | h_{i-M+1}, \ldots, h_{i-1})$ is the parameter of this generative model. We denote the set of all $\Pr(h_i | h_{i-M+1}, \ldots, h_{i-1})$ as $\mathbf{w}$.

To estimate the parameters for this generative model, we use the expectation-maximization (EM) algorithm. We would like to find the parameter that maximize the log likelihood of the training set D:

$$\begin{aligned} \ell(\mathbf{w}) &= \sum_{(x,y) \in D} \log \Pr(x, y) \\ &= \sum_{(x,y) \in D} \log \sum_{h \in \mathbf{H}(x,y)} \Pr(h). \end{aligned} \tag{4.11}$$

To find the optimal parameter $\mathbf{w}$, EM algorithm iteratively does a series of:

1. expectation step that calculates the soft counts $c(h_{i-M+1}, \ldots, h_i)$, which is the count of M-gram $h_{i-M+1}, \ldots, h_i$ weighted by the probability of the latent label h in which it occurs.

$$\begin{aligned} c(h_{i-M+1}, \ldots, h_i) &= \sum_{(x,y) \in D} \sum_{h \in \mathbf{H}(x,y)} \Pr(h | x, y) n_{h_{i-M+1}, \ldots, h_i}(h) \\ &= \sum_{(x,y) \in D} \sum_{h \in \mathbf{H}(x,y)} \frac{\Pr(h)}{\sum_{h' \in \mathbf{H}(x,y)} \Pr(h')} n_{h_{i-M+1}, \ldots, h_i}(h), \end{aligned} \tag{4.12}$$

   where $n_{h_{i-M+1}, \ldots, h_i}(h)$ is the number of occurrences of M-gram $h_{i-M+1}, \ldots, h_i$ in latent label h.

2. maximization step that maximizes the likelihood of the training set given the soft counts:

$$\Pr(h_i|h_{i-M+1}\dots h_{i-1}) = \frac{c(h_{i-M+1},\dots,h_i)}{\sum_{h'_{i-M+1},\dots,h'_{i-1}} c(h'_{i-M+1},\dots,h'_{i-1},h_i)}$$

Theoretically there is no guarantee that this EM algorithm converges to the optimal since the log likelihood objective (Equation 4.11) is not convex.

Furthermore, consider the expectation step (Equation 4.12) where for each word pronunciation pair $(x, y) \in D$ in the training set, EM needs to iterate through all possible latent labels $h \in \mathbf{H}(x, y)$. This search space is of size exponential to the length of x and y and the exact search is not affordable, so in practice we resort to inexact search like beam search.

Chapter 5: Latent Variable Structured Learning: Applications

Here we show some applications of latent variable structured learning. We show that the strong generalizing capability makes latent variable structured learning a very powerful model in practice.

5.1 Incremental Semantic Parsing

As we discussed in Chapter 1.3, for many NLP tasks it is easier to annotate partial output (natural annotation), but not the whole output.

Our first application for latent variable structured learning is incremental semantic parsing.

There are two types of natural annotations for semantic parsing task.

1. Annotation of meaning representations. This is the most straightforward annotation: for each sentence, the corresponding meaning representation is annotated. There are several different choices for meaning representation: the first-order logic form [51], the λ expression form with first order logic [64], and λ dependency-based compositional semantics [32].

2. Annotation of question answers which annotates the answer to each question while the meaning representation for the question is hidden [28, 5].

Here we focus on the first type of natural annotation with λ expression with first order logic as the meaning representation, and use latent variable structured perceptron as the statistical model.

5.1.1 Incremental Semantic Parsing Decoding

We use the Type-driven Incremental Semantic Parsing (Tisp) algorithm [67] to demonstrate the decoding process of incremental semantic parsing. We use the following running example in the domain of GeoQuery dataset:

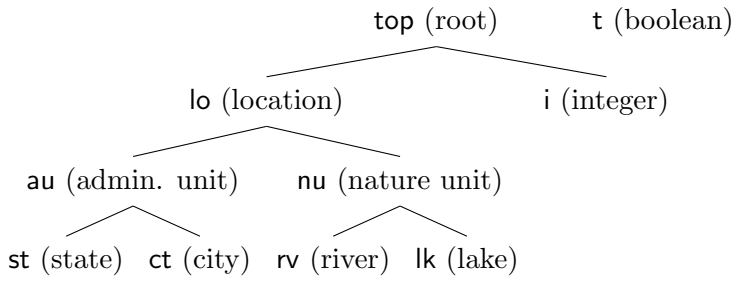


Figure 5.1: Type hierarchy for GeoQuery domain (slightly simplified for presentation).

Q: What is the capital of the largest state by area?

MR: (**capital** (**argmax** **state** **size**)) : ct

Note that the meaning representation is annotated with its type. In the above example the meaning representation is of type city (ct). Figure 5.1 shows the subtype hierarchy used in the λ expression meaning representation.

Similar to incremental (shift-reduce) dependency parsing, Tisp maintains a *stack* and a *queue* during the parsing. The stack is initialized to be empty, while the queue is initialized to be the sentence to be parsed with the first word as the head of the queue.

At each parsing step, Tisp choose to perform one of following three actions:

1. **shift** (sh): Tisp pops one word from the queue, finds a predicate or constant defined in the domain that matches the word to form a subexpression, and pushes the subexpression onto the stack.

2. **reduce** (re): Tisp pops the top two subsexpressions s_0 and s_1 from the stack, and checks whether they are applicable/reducible. If s_0 and s_1 are reducible[1], Tisp performs the function application and pushes the result subexpression onto the stack.

3. **skip** (sk): Tisp pops the top word from the queue, and ignores it. This step is mainly to skip those semantically vacuous words.

Table 5.1 shows an exemplary parsing derivation for Tisp. There are several issues we would like to address in the decoding procedure.

[1] There can be only one way to reduce them.

step	action	stack after action	queue	typing
0	-	ϕ	what...	
1–3	sk	ϕ	capital...	
4	$\mathsf{sh}_{\mathsf{capital}}$	**capital**:st → ct	of...	
7	$\mathsf{sh}_{\mathsf{largest}}$	**capital**:st → ct **argmax**:('a → t) → ('a → i) → 'a	state...	
8	$\mathsf{sh}_{\mathsf{state}}$	**capital**:st → ct **argmax**:('a → t) → ('a → i) → 'a **state**:st → t	by...	
9	$\mathsf{re}_{\curvearrowright}$	**capital**:st → ct (**argmax state**):(st → i) → st	by...	binding: 'a = st
11	sh area	**capital**:st → ct (**argmax state**):(st → i) → st **size**:lo → i	?	
12	$\mathsf{re}_{\curvearrowright}$	**capital**:st → ct (**argmax state size**):st	?	st <: lo ⇒ (lo → i) <:(st → i)
13	$\mathsf{re}_{\curvearrowright}$	(**capital** (**argmax state size**)):ct	?	

Table 5.1: Exemplary parsing derivation for the running example in Tisp.

pattern	λ-expression templates with polymorphic types
JJS	λP:('a → t) → ('a → i) → 'a . P
NN	λP: 'b → 'c . P

Table 5.2: POS-based meaning representation templates used in the running example (Table 5.1). .

First, for the shift action, once a word is popped from the queue, Tisp needs to find the best predicate or constant matching this word to be pushed onto the stack. To figure out which predicate or constant is favorable, Tisp first finds the corresponding λ-expression templates based on the POS tag of the word, and then it instantiates the templates using predicates that matches the templates in types. For example, the λ-expression templates used for the exemplary parsing derivation (Table 5.1) are listed in Table 5.2. In step 4, when the word "capital" is shifted, based on its POS tag NN, template λP : 'b → 'c . P is chosen. Among all those predicates in the database, **capital** : st → ct matches this template and, thus, is used to instantiates the template to get **capital** : st → ct to be pushed onto the stack.

Second, for the reduce action where Tisp applies function application, the type system requires the type of the argument to be subtype of the type of the function's argument. More formally we have

$$\frac{e_2 : \mathsf{S} \qquad \mathsf{S} <: \mathsf{T}}{(\lambda x : \mathsf{T} \,.\, e_1)\ \ e_2\ \ \rightarrow\ \ [x \mapsto e_2]e_1},$$

where $[x \mapsto e_2]$ means replace the occurrence of variable x in expression e_1 with expres-

sion e_2. One example of the typed function application is in step 13 where the function is **capital** : st $\rightarrow$ ct, and the argument is (**argmax state size**):st, which is applicable to the function since st is a subtype of itself: st <: st. One tricky example is in step 12 where the function is of type (st $\rightarrow$ i) $\rightarrow$ st and the argument is of type lo $\rightarrow$ i. To make this application sound in types, we need (lo $\rightarrow$ i) <: (st $\rightarrow$ i) given that st <: lo in Figure 5.1. This seems counter-intuitive at first, but it is reasonable since the function would require its argument (which is a function of lo $\rightarrow$ i) to not be surprised by any possible future arguments (which are at most st). More formally this is defined as a contravariant rule in type theory:

$$\frac{\mathsf{A} <: \mathsf{B}}{\mathsf{B} \rightarrow \mathsf{C} <: \mathsf{A} \rightarrow \mathsf{C}}.$$

Third, besides the types in Figure 5.1, TISP also uses type variables. Consider the type of the return value of **argmax**: in questions like "what is the largest river" it should return value of type rv, while in questions like "what is the largest city" it should return value of type ct. This means the correct return type of **argmax** can not be determined at the time when it is pushed onto the stack. Instead its return type is flexible based on its arguments. We model this flexibility with type variables in the type theory. One example is in step 9 of Table 5.1 where the return type of **argmax** is determined by its argument **state** : st, which can be formally written as type variable 'a being bound to st: 'a = st.

The constraints from the subtype hierarchy and function application can avoid some unnecessary function applications in the search for the correct parses. However, spurious ambiguity like Table 2.1 still can not be totally avoided.[2] This spurious ambiguity can be caused by how the templates are chosen and grounded in the shift step, and the different reduce orders that lead to the same result. We treat this ambiguity as latent variables and handle the learning problem with latent variable structured perceptron.

[2] The spurious ambiguity is more severe in semantic parsing with only annotated question-answer pairs where even the parse tree is hidden and only the evaluation result of the parse tree is given. Different parse trees can be evaluated to give the same answer.

5.1.2 Latent Variable Structured Perceptron for Incremental Semantic Parsing

Due to its similarity to the phrase-based machine translation task, we model the incremental semantic parsing task similar to the latent variable structured perceptron learning for phrase-based machine translation [62].

To perform a structured perceptron update to correct a search error, we need to consider two issues:

1. Different from our POS tagging example for structured perceptron without latent variable, there can be multiple correct latent derivation that lead to the correct output. This issue has been addressed by Equation 4.7.

2. However, trivially applying Equation 4.7 is problematic since it does not consider the problem of inexact search. For the problem of incremental semantic parsing, beam search is also incorporated for decoding efficiency. The decoding process of incremental semantic parsing is similar to the POS tagging task where a derivation is a sequence, so the "early update" and "max-violation update" strategies defined in Figure 3.2 should be adapted to handle latent variables.

More formally for a meaning representation pair (x, y), we denote $D(x)$ to be the set of *all* partial and full parsing derivations for input sentence x. For a full derivation $d \in D(x)$, we define $mr(x)$ to be the meaning representation of d. We then can define the set of (partial and full) reference derivations at step i to be:

$$good_i(x, y) \stackrel{\Delta}{=} \{d \in D(x) \mid |d| = i, \exists \text{full derivation } d' \text{ s.t. } d \text{ is a prefix of } d', mr(d') = y\},$$

We can also define the set of (partial and full) bad derivations at step i to be:

$$bad_i(x, y) \stackrel{\Delta}{=} \{d \in D(x) \mid d \notin good_i(x, y), |d| = i\}.$$

At step i, the best reference (partial or full) derivation is

$$d_i^+(x, y) \stackrel{\Delta}{=} \underset{d \in good_i(x,y)}{\text{argmax}} \ \mathbf{w} \cdot \mathbf{\Phi}(x, d).$$

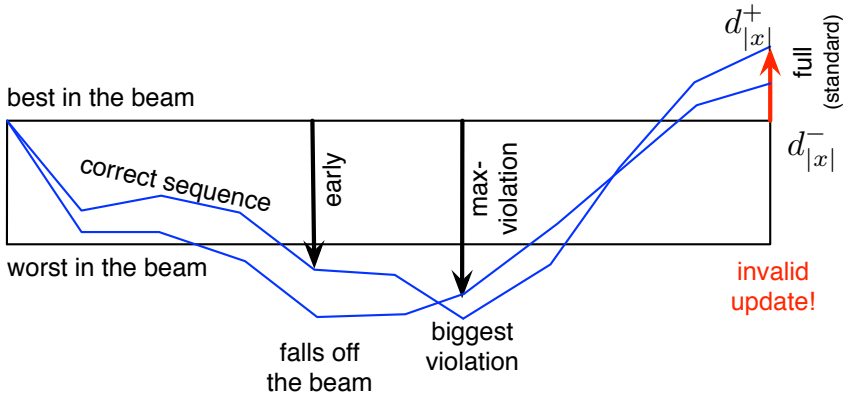


Figure 5.2: Update methods for latent variable structured perceptron with beam search.

In practice, searching over the space of all possible derivations $D(x)$ is very time-consuming and not affordable. So as usual we resort to inexact search, i.e., beam search. We define the set of (partial and full) derivations explored in the beam search as $\tilde{D}(x) \subseteq D(x)$. The incorrect Viterbi (partial or full) derivation at step i is

$$d_i^-(x,y) \stackrel{\Delta}{=} \operatorname*{argmax}_{d \in bad_i(x,y) \cap \tilde{D}(x)} \mathbf{w} \cdot \mathbf{\Phi}(x,d),$$

Following [62], we can find the step i^* where the difference between the score of the best reference derivation and the score of the incorrect Viterbi derivation is maximal:

$$i^* \stackrel{\Delta}{=} \operatorname*{argmax}_i \mathbf{w} \cdot \mathbf{\Delta\Phi}(x, d_i^+(x,y), d_i^-(x,y)),$$

and do update at step i^*:

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{\Delta\Phi}(x, d_{i^*}^+(x,y), d_{i^*}^-(x,y)),$$

where $\mathbf{\Delta\Phi}(x,d,d') \stackrel{\Delta}{=} \mathbf{\Phi}(x,d) - \mathbf{\Phi}(x,d')$.

Based on this definition, we can draw the update strategies in Figure 5.2, which is corresponding to Figure 3.2 for the structured perceptron with beam search without latent variables.

System	GeoQuery			Jobs			Atis		
	P	R	F1	P	R	F1	P	R	F1
Z&C'05 [64]	**96.3**	79.3	87.0	**97.3**	79.3	87.4	-	-	-
Z&C'07 [65]	91.6	86.1	88.8	-	-	-	**85.8**	**84.6**	**85.2**
UBL [29]	94.1	85.0	89.3	-	-	-	72.1	71.4	71.7
FUBL [30]	88.6	88.6	88.6	-	-	-	82.8	82.8	82.8
Tisp (st)	89.7	86.8	88.2	76.4	76.4	76.4	80.7	80.4	80.5
Tisp	92.9	**88.9**	**90.9**	85.0	**85.0**	**85.0**	84.7	84.2	84.4

Table 5.3: Performances (precision, recall, and F1) of various parsing algorithms on GeoQuery, Jobs, and Atis datasets. Tisp with simple types are marked "st".

5.1.3 Empirical Evaluation

We evaluate the latent variable structured perceptron with inexact search over three standard evaluation sets: GeoQuery, Jobs, and Atis.

Our feature set consists of Word-Edge features that have been shown to be effective in constituent parsing [10] and machine translation [62]. In detail we include features of combinations of the types and leftmost and rightmost words of the top 3 meaning representations on the stack, the top 3 words on the queue, the grounded predicate names and the rules being used. To avoid overfitting problem due to very complex features, we set a budget to each type of atomic feature and only feature combinations below a fixed budget threshold are used.

We evaluate the performance of Tisp with definitions of *precision* and *recall* specially tailored for semantic parsing task, which have widely being used in other semantic parsing approaches.

$$\text{Precision} = \frac{\text{\# of correctly parsed questions}}{\text{\# of successfully parsed questions}},$$

$$\text{Recall} = \frac{\text{\# of correctly parsed questions}}{\text{\# of questions}}.$$

We show the performance comparison in Table 5.3. In general, Tisp can achieve competitive performance in all three datasets.

We also run ablation experiments where the subtype hierarchies for the three tasks are simplified to contain only entity (`e`), integer (`i`), and boolean (`t`). The results are also shown in Table 5.3. From these ablation experiments we can see that the refined subtype hierarchies can indeed improve the search by avoiding unnecessary search branches, and,

id	rule
r_0	$\mathrm{S} \rightarrow \langle \mathrm{X}_{\boxed{1}}, \mathrm{X}_{\boxed{1}} \rangle$
r_1	$\mathrm{S} \rightarrow \langle \mathrm{S}_{\boxed{1}} \mathrm{X}_{\boxed{2}}, \mathrm{S}_{\boxed{1}} \mathrm{X}_{\boxed{2}} \rangle$
r_2	$\mathrm{X} \rightarrow \langle \textit{Bùshí}, \text{ Bush} \rangle$
r_3	$\mathrm{X} \rightarrow \langle \textit{Shālóng}, \text{ Sharon} \rangle$
r_4	$\mathrm{X} \rightarrow \langle \textit{huìtán}, \text{ talks} \rangle$
r_5	$\mathrm{X} \rightarrow \langle \textit{yǔ } \mathrm{X}_{\boxed{1}} \textit{ jǔxíng } \mathrm{X}_{\boxed{2}}, \text{ held } \mathrm{X}_{\boxed{2}} \text{ with } \mathrm{X}_{\boxed{1}} \rangle$
r_6	$\mathrm{X} \rightarrow \langle \textit{yǔ Shālóng}, \text{ with Sharon} \rangle$
r_7	$\mathrm{X} \rightarrow \langle \mathrm{X}_{\boxed{1}} \textit{ jǔxíng } \mathrm{X}_{\boxed{2}}, \mathrm{X}_{\boxed{1}} \text{ held } \mathrm{X}_{\boxed{2}} \rangle$

Table 5.4: Exemplary HIERO translation rules. Rules r_0 and r_1 are glue rules.

thus, improve the search quality.

5.2 Syntax-base Machine Translation

Our second application for latent variable structured learning is syntax-based machine translation [59, 12]. Syntax-based machine translation contains several different approaches. Here we focus on one special approach: the hierarchical phrase-based translation model (HIERO) [12].

5.2.1 HIERO Decoding

Here we first briefly describe the HIERO decoding algorithm, which can be viewed as a two-step process.

In the first step, HIERO parses following the HIERO synchronous grammar. Table 5.4 shows an exemplary set of translation rules for Chinese-English Translation. As a synchronous grammar, the right hand side of each HIERO rule is a pair describing both the source sentence and the target sentence. In the source side or the target side, there can be at most two non-terminals. There are two special rules, r_1 and r_2 in Table 5.4, that are called glue rules which are used to convert non-terminal X to the root non-terminal S and to concatenate the translations from non-terminals respectively.

In this step, HIERO decoder parses the source sentence using the source sides of the translation rules. Exemplary source side parse trees are shown as the top trees in Figure 5.3. In the mean time, HIERO generates the target sentence using the target sides

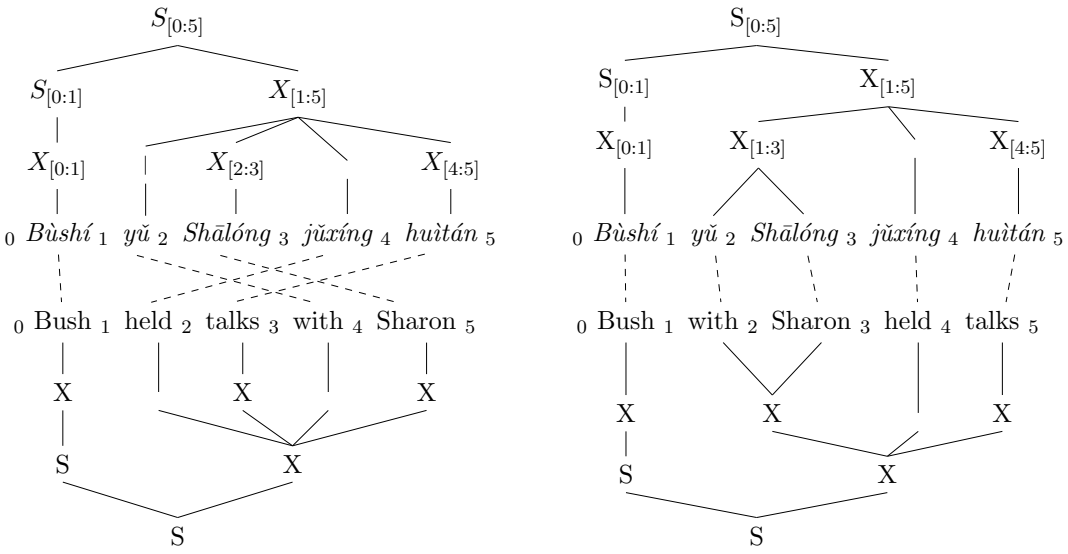


Figure 5.3: An example of HIERO translation. (a)–(b): gold and Viterbi derivations of HIERO translation represented by synchronous trees respectively. The top trees are in source side and the bottom trees are in target side. The subscript of a node in the source tree shows the source side span of that node. The dashed lines show the alignment between source and target words.

of the translation rules. The bottom trees in Figure 5.3 are the corresponding target parse trees generated from the source parses using rules in Table 5.4.

More formally, the result source parses can be represented as a *hypergraph*, which is also called a *forest* since a hypergraph is an aggregated representation of a group of trees. In this hypergraph, each node has a signature of form $N_{[i:j]}$, where N, being either S or X in HIERO, is the non-terminal of the left hand side of the rule used to generate the subtrees rooted at span $[i:j]$. Each hyperedge e in the hypergraph is an application of the translation rule $r(e)$.

Figure 5.4 shows an exemplary hypergraph containing both of the source parses in Figure 5.3.

From the target tree generated in the first step, we can easily generate the translated sentence. However, this sentence is usually not fluent enough. To overcome this problem, we explicitly add a score from a language model to rank the rule combinations that

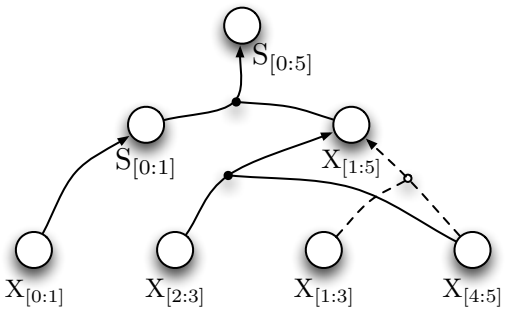


Figure 5.4: A hypergraph containing two derivations: the gold derivation (Figure 5.3 (a)) in solid lines, and the Viterbi derivation (Figure 5.3 (b)) in dashed lines.

produce fluent translation. This is done in the second step of HIERO decoding.

More formally, the second step augment the above hypergraph by adding an extra mark of language model for each hypergraph node. After the augmentation, each node $N_{[i:j]}$ in the hypergraph is split into multiple nodes of signature $N^{a\star b}_{[i:j]}$, where a and b are translated boundary words of span $[i:j]$. For bigram language model, both a and b contain one word, since it will be enough to calculate the language model when two spans are concatenated. One example of the augmented node with bigram language model for the Viterbi derivation in Figure 5.3 is $\mathrm{X}^{\mathrm{with}\star\mathrm{talks}}_{[1:5]}$ since the translation of the node $X_{[1:5]}$ in the Viterbi derivation is "with Sharon held talks" whose boundaries are "with" and "talks".

Actually we can represent the HIERO decoding process formally as a deductive system. Take the partial translation "with Sharon held taks" in the Viterbi derivation of Figure 5.3 (b) as an example, which is the result of applying rule r_7 in Table 5.4 over nodes $\mathrm{X}^{\mathrm{with}\star\mathrm{Sharon}}_{[1:3]}$ and $\mathrm{X}^{\mathrm{talks}\star\mathrm{talks}}_{[4:5]}$, the corresponding deduction is:

$$\frac{\mathrm{X}^{\mathrm{with}\star\mathrm{Sharon}}_{[1:3]} : s_1 \qquad \mathrm{X}^{\mathrm{talks}\star\mathrm{talks}}_{[4:5]} : s_2}{\mathrm{X}^{\mathrm{with}\star\mathrm{talks}}_{[1:5]} : s_1 + s_2 + s(r_7) + \lambda} \quad r_7, \tag{5.1}$$

where s_1 and s_2 are the scores for the derivations at nodes $\mathrm{X}^{\mathrm{with}\star\mathrm{Sharon}}_{[1:3]}$ and $\mathrm{X}^{\mathrm{talks}\star\mathrm{talks}}_{[4:5]}$ respectively, $s(r_7)$ is the score from applying translation rule r_7, and λ is the language model score which for this example is $\log \mathrm{Pr}_{\mathrm{LM}}(\mathrm{held}|\mathrm{Sharon})\mathrm{Pr}_{\mathrm{LM}}(\mathrm{talks}|\mathrm{held})$.

5.2.2 Latent Variable Structured Perceptron for HIERO

Consider the application of structured perceptron for this problem. As we mentioned before, fully annotating the parse hypergraph for each machine translation example requires experts and is very time-consuming. Instead most conventional machine translation systems are trained on parallel sentence pairs.

For a parallel sentence pair like Figure 1.4, only the correct translation for the source sentence is annotated. We can only confirm the correctness of a translation derivation if its translated sentence agrees with the provided reference.

To perform a structured perceptron update to correct a search error for HIERO, we need to adapt it to the hypergraph search setting. The decoding process of incremental semantic parsing is similar to the POS tagging task where a derivation is a sequence, so the "early update" and "max-violation update" strategies defined in Figure 3.2 can be easily applied. For HIERO decoding the derivation is of tree structure, for which the "early update" and "max-violation update" strategies are not yet defined. Fortunately, applying perceptron update in the middle of the search where the derivations are trees for bottom-up parsing has been investigated in [66], which resembles HIERO decoding. We use similar approaches as [66] but take the latent variables into consideration.

More formally, consider a sentence pair $\langle x, y \rangle$, we denote $H(x)$ as the HIERO decoding hypergraph. We denote $D \in H(x)$ if D is a full derivation generated in the decoding of x, and D belongs to the hypergraph $H(x)$. We say D is a correct derivation if its translated sentence agrees with the reference sentence y. Note that there can be more than one full derivations that are correct. We denote the set of correct full derivations, or *good* derivations as:

$$good(x, y) \triangleq \{D \in H(x) \mid e(D) = y\},$$

where $e(D)$ is the English side, i.e., the translated sentence, of derivation D.

We can also define the partial good translations that cover span $[i : j]$ with node $N_{[i:j]}$ in the hypergraph:

$$good_{N_{[i:j]}}(x, y) \triangleq \{d \in D \mid D \in good(x, y), root(d) = N_{[i:j]}\} \tag{5.2}$$

However, in real decoding time, due to the exponentially large search space, it is im-

practical to search over the whole derivation hypergraph. We usually resort to inexact search, i.e., beam search. Furthermore, in practice the searching time with only beam search is not affordable due to the huge amount of translation rules and different combinations of phrases for language model evaluation. Cube-pruning [22] is widely used to further speedup the searching by introducing more approximation.[3]

This means that the real decoding hypergraph $\tilde{H}(x)$ is a subset of the full decoding hypergraph $H(x)$. We denote the wrong (partial) derivations generated during the real decoding time in hypergraph $\tilde{H}(x)$ as *bad* (partial) derivations:

$$bad_{N_{[i:j]}}(x,y) \triangleq \{d \in D \mid D \in \tilde{H}(x,y), root(d) = N_{[i:j]}, d \notin good_{N_{[i:j]}}(x,y)\}.$$

Note that the y-good derivations are defined over the full decoding hypergraph $H(x)$ while the y-bad derivations are defined over the real decoding hypergraph $\tilde{H}(x)$. They are not defined symmetrically.

Following the definition of "max-violation update" in [23, 62, 66], we define our "max-violation update" at the hypergraph where the score difference between the incorrect Viterbi partial derivation and the best reference partial derivation is *maximal*.

In detail, we first find the reference partial derivation with the highest score, d^+, at each hypergraph node $N_{[i:j]}$ in the whole hypergraph $H(x)$:

$$d^+_{N_{[i:j]}}(x,y) \triangleq \operatorname*{argmax}_{d \in good_{N_{[i:j]}}(x,y)} \mathbf{w} \cdot \mathbf{\Phi}(x,d),$$

and the incorrect Viterbi partial derivation d^- at each hypergraph node $N_{[i:j]}$ in the decoding hypergraph $\tilde{H}(x)$:

$$d^-_{N_{[i:j]}}(x,y) \triangleq \operatorname*{argmax}_{d \in bad_{N_{[i:j]}}(x,y)} \mathbf{w} \cdot \mathbf{\Phi}(x,d).$$

Then we find the node $N^*_{[i^*:j^*]}$ where the score difference between the previously found

[3] Actually this extra approximation can also be problematic for the convergence proof for structured perceptron since it can be another source of violation. But here we skip the discussion of cube-pruning for simplicity reason.

best reference partial derivation and the Viterbi partial derivation is maximal:

$$N^*_{[i^*:j^*]} \triangleq \operatorname*{argmax}_{N_{[i:j]}} \mathbf{w} \cdot \Delta\mathbf{\Phi}(x, d^+_{N_{[i:j]}}(x,y), d^-_{N_{[i:j]}}(x,y)),$$

and do one max-violation update:

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{\Phi}(x, d^+_{N^*_{[i^*:j^*]}}(x,y), d^-_{N^*_{[i^*:j^*]}}(x,y)),$$

where $\Delta\mathbf{\Phi}(x,d,d') \triangleq \mathbf{\Phi}(x,d) - \mathbf{\Phi}(x,d')$.

One issue remains in above definition. Consider the definition in Equation 5.2, where the reference partial derivations are defined on the whole decoding hypergraph. This is impractical due to the huge search space.

To efficiently find the correct partial derivations, we do a *forced decoding* process which essential implements the definition of Equation 4.3.

We do forced decoding by using a specially designed language model in the second step of HIERO decoding:

$$\Pr_{forced}(q \mid p) = \begin{cases} 1 & \text{if } q = p+1 \\ 0 & \text{otherwise} \end{cases},$$

where p and q are the indices of the boundary words in the reference translation.

With this language model, the nodes in the new decoding hypergraph are of form $N^{p\star q}_{[i:j]}$. If a boundary word does not occur in the reference translation, its index is set to ∞ so that the corresponding language model score will always be $-\infty$.

The decoding deductive system now can be adapted for forced decoding. Rule r_7 in Table 5.4 can be written as:

$$\mathrm{X} \rightarrow \langle \mathrm{X}_{\boxed{1}} \textit{ jǔxíng } \mathrm{X}_{\boxed{2}},\ \mathrm{X}_{\boxed{1}}\ 1\ \mathrm{X}_{\boxed{2}} \rangle,$$

The deductive step corresponding to Equation 5.1 now can be written as:

$$\frac{\mathrm{X}^{3\star4}_{[1:3]} : s_1 \qquad \mathrm{X}^{2\star2}_{[4:5]} : s_2}{\mathrm{X}^{4\star2}_{[1:5]} : s_1 + s_2 + s(r_7) + \lambda} \quad r_7,$$

where $\lambda = \log \Pr_{forced}(1|4)\Pr_{forced}(2|1) = -\infty$, which will be pruned since it does not lead to reference translation.

5.2.3 Empirical Evaluation

Following [62] we call our latent variable structured perceptron with inexact search for syntax-based machine translation MaxForce. We evaluate this proposed method in [69] over two Chinese-English corpora, IWSLT09 and FBIS. Our implementation is based on `cdec` [19] with its `pycdec` [9] interface. We compare this method with conventional parameter tuning methods including n-best Mert [40], hypergraph Mert [27], and Pro [21].

We design the features for structured perceptron containing two sets

1. All the dense features widely used in all conventional parameter tuning methods including language model score, direct translation probability $p(e|f)$, lexical translation probabilities $p_{\text{LEX}}(e|f)$ and $p_{\text{LEX}}(f|e)$, length penalty, counts for the source and target phrases in the training corpus, and flags for the glue rules and pass-through rules.

2. For sparse features we use Word-Edge features which are very effective in parsing [10] and phrase-based MT [62]. Our Word-Edge features consist of combinations of English and Chinese words, and Chinese characters on the boundary of the translation rules. To avoid overfitting problem from complex features, we apply a budget scheme that only simple feature combinations within the given budget are used.

We first evaluate the translation result over the two datasets with Bleu score [41]. Table 5.5 and Table 5.6 show the comparisons between MaxForce and other conventional methods. In addition, we also compare MaxForce with the "local update" method of [33] which update by rewarding the derivation at the root span $S_{0:|x|}$ with the highest sentence-level Bleu score. The result from local update is ~ 2 Bleu worse than MaxForce.

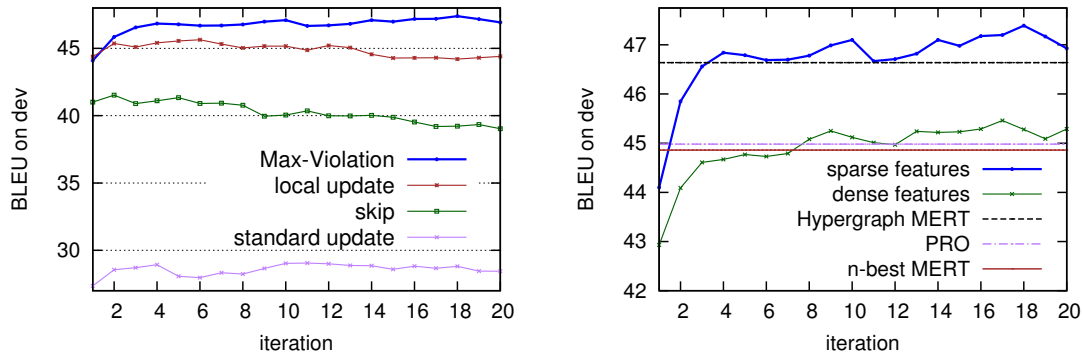
We also run two ablation experiments to evaluate the components of MaxForce training.

algorithm	# feats	dev	test
n-best MERT	18	44.9	47.9
Hypergraph MERT	18	46.6	50.7
PRO	18	45.0	49.5
local update perc.	443K	45.6	49.1
MAXFORCE	529K	**47.4**	**51.5**

Table 5.5: BLEU scores (with 16 references) of various training algorithms on IWSLT09. IWSLT04 is used as tuning set for MERT and PRO, and as development set for MAXFORCE and local update perceptron of [33]; IWSLT05 is used as test set.

algorithm	# feats	dev	test
Hypergraph MERT	18	27.3	23.0
PRO	18	26.4	22.7
MAXFORCE	4.5M	**27.7**	**23.9**

Table 5.6: BLEU scores (with 4 references) of various training algorithms on FBIS. NIST06 newswire is used as tuning set for Hypergraph MERT and PRO, and as development set for MAXFORCE; NIST08 newswire is used as test set.



(a) (b)

Figure 5.5: (a) Comparison of various update methods over IWSLT09 for latent variable structured perceptron. (b) For IWSLT09, sparse features contribute ~ 2 BLEU to MAXFORCE, outperforming MERT and PRO.

The first ablation experiment (Figure 5.5 (a)) compares MaxForce with different update strategies: max-violation [23], local update [33], skip [66] which updates at the root span only when the update can fix a search error, and standard update. The difference between standard update and all other update methods is significant ($\sim$ 10 Bleu) since standard update does not have any convergence guarantee as we discussed in Chapter 3.1.3. The difference between skip strategy and the other two update strategies is $\sim$ 5 Bleu, which can be explained by the fact that there are fewer updates with skip strategies than the other two.

The other ablation experiment (Figure 5.5 (b)) evaluates the contributions of different features in MaxForce. We find that sparse features contribute $\sim$ 2 Bleu to MaxForce, making it outperform Mert and Pro. With only dense features MaxForce is roughly of the same performance as Mert and Pro.

5.3 Sentence Entailment

Besides semantic parsing and machine translation, we would also like to evaluate the latent variable structured learning technique in the application of sentence entailment, together with the neural network model [68].

Recall the sentence entailment problem we discuss in Figure 1.5 where the goal is to determine whether a sentence (hypothesis) can be inferred from another given sentence (premise). There are two general approaches to solve this problem.

1. The first approach uses symbolic inference rules to check whether the hypothesis can be induced from the premise with logic. A typical example is natural logic [34] where the premise is converted to the hypothesis with a series of sentence edits of insertion, replacement, and deletion of words. Each sentence edit has a corresponding logic operation, and the final entailment relation can be calculated by composing these logic operations together following some composition rules. Another example is based on semantic parsing [53], where both of the premise and the hypothesis are first converted to semantic parse trees, and then a series of logic operations specially designed for semantic parses trees are applied to check if there is valid entailment relation.

   The symbolic inference approaches are vulnerable to the data sparsity problem

since their underlying models, being either structured perceptron or CRF, depend on sparse features, which means they can handle those sentences occur in the training set, but does not generalize well with unseen sentences in the development and testing sets. This problem is even more severe in sentence entailment task since usually the training sets for sentence entailment are small.

2. Another approach is based on the strong generalization power of the continuous representations, i.e., word embedding, and neural networks. One exemplary attempt for solving sentence entailment problem with neural models is [8], where both the premise and the hypothesis are passed into two neural networks separately to get two continuous representations of the meaning of the sentences. After that these two vectors are passed into another neural network that determine the relationship between the two sentences. This model shows that with sufficiently large training set, continuous representations can generalize well and outperform discrete symbolic models. Another example is [45] where an attention model is introduced to determine for each word in the hypothesis, which word in the premise is most important in determining the entailment relationship.

   The problem with these neural models is that they either ignore the latent entailment relation in the between parts of the sentences of the premise and the hypothesis [8], or they simply model this relation in a word-by-word way, instead of the more reasonable tree node-level alignment as in Figure 1.5.

We would like to leverage the structure of the latent alignment to further improve the sentence entailment performance. This means the node-level attention should follow the constraints from the structure of the two parses trees, and also the composition of the entailment relations should happen at the tree node level, rather than at the sentence level.

5.3.1 Formalization

We first formalize the sentence entailment problem as a structured prediction problem similar to [38, 2, 60]. The input for the sentence entailment task are two trees: the premise tree P and the hypothesis tree Q. We assume both trees are binarized. The goal of the sentence entailment task is to predict a label $y \in \{\text{entailment}, \text{neutral}, \text{contradiction}\}$.

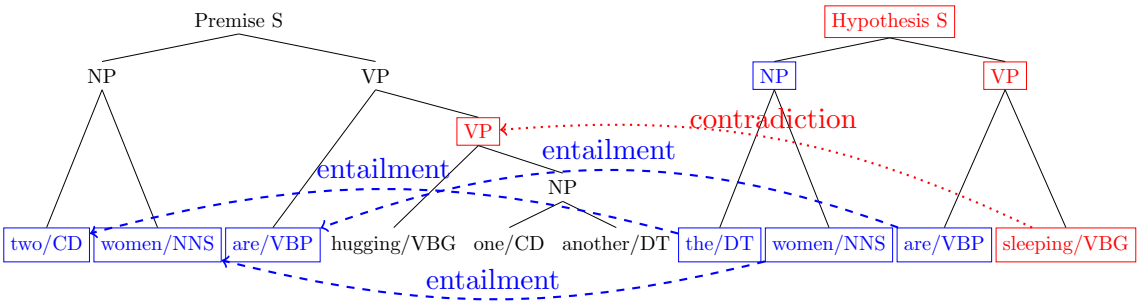


Figure 5.6: Exemplary trees for the premise sentence "two women are hugging one another" and the hypothesis sentence "the women are sleeping". The syntactic labels (NP, VP, CD, etc.) are not used in the model. The dashed and dotted lines show the lowest level of alignments from the hypothesis tree nodes to the premise tree nodes. The blue dashed lines mark the entailment relations, and the red dotted line marks the contradiction relation. In the hypothesis tree, tree nodes in blue squares are identified to be entailment from the premise, and nodes in red squared are identified to contradicts the premise. By composing these relations from the bottom up, we reach a conclusion that the sentence-level entailment relation is contradiction. Please also refer to Figure 5.11 for real examples taken from our experiments.

Note that at first glance, this problem is not a structured prediction problem: the output is a label rather than a structure such as a sequence or a tree. But here we still consider this problem as a structured prediction problem since: 1) the input to the problem is structured; 2) the approach we choose to analyze the input relies on internal structures, i.e., alignments.

5.3.1.1 Decomposition of the Problem

We can formally write down the goal of the decoding algorithm as to find the label with the minimal negative log likelihood, given the input pair of trees:

$$\operatorname*{argmin}_y -\log \Pr(y|P,Q)$$

Then we introduce the internal alignments $\mathbf{A} \in \{0,1\}^{|Q|\times|P|}$ between tree nodes, where $|\cdot|$ calculates the number of nodes in a tree. $\mathbf{A}_{ij} = 1$ if and only if node i in hypothesis tree Q is aligned to node j in premise tree P.

$$\begin{aligned}
&\operatorname*{argmin}_y -\log \Pr(y|P,Q) \\
=&\operatorname*{argmin}_y -\log \sum_{\mathbf{A}} \Pr(y,\mathbf{A}|P,Q) \\
=&\operatorname*{argmin}_y -\log \sum_{\mathbf{A}} \Pr(\mathbf{A}|P,Q)\cdot\Pr(y|\mathbf{A},P,Q) \\
=&\operatorname*{argmin}_y -\log \mathbb{E}_{\Pr(\mathbf{A}|P,Q)}[\Pr(y|\mathbf{A},P,Q)]
\end{aligned} \tag{5.3}$$

We now have two separate problems to discuss:

- Given the input pair of trees, we need to calculate the probability for alignment $\mathbf{A}$:

$$\Pr(\mathbf{A}|P,Q).$$

- Given a fixed alignment $\mathbf{A}$, we need to infer the probability of the final label y:

$$\Pr(y|\mathbf{A},P,Q).$$

5.3.1.2 Architecture of the Network

Here we first briefly introduce how to construct neural networks to solve the above two problems separately, and later in the next subsection we will discuss how to jointly train the model with the alignment $\mathbf{A}$ as a latent variable.

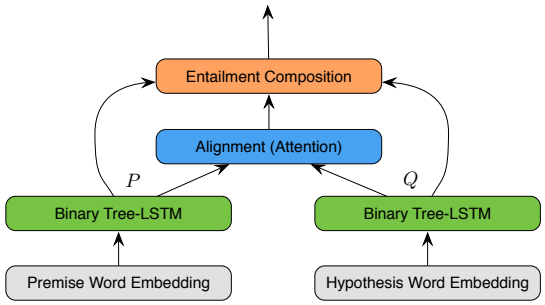


Figure 5.7: Network architecture for Sentence Entailment task.

As Figure 5.7 shows, we use two networks to compute the alignment and the overall entailment separately. We assume that there is a meaning representation $\mathbf{h}$ computed for each node in the premise and hypothesis trees, which we will address later in this subsection.

First, for the alignment, we approximate the global (binary) alignment $\mathbf{A}$ to be consisted of the alignment $\mathbf{A}_i \in \{0,1\}^{1\times|P|}$ of each node in the hypothesis tree Q independently:

$$\mathbf{A} = [\mathbf{A}_1^T; \mathbf{A}_2^T; \ldots; \mathbf{A}_{|Q|}^T]^T,$$

$$\Pr(\mathbf{A}|P,Q) = \prod_i^{|Q|} \Pr(\mathbf{A}_i|P,Q).$$

$\Pr(\mathbf{A}_{i,j} = 1|P,Q)$ is the probability of the node $i \in Q$ being aligned to node $j \in P$, which is defined as

$$\Pr(\mathbf{A}_{i,j} = 1|P,Q) \triangleq \frac{\exp(T_{2k,1}([\mathbf{h}_i;\mathbf{h}_j]))}{\sum_t \exp(T_{2k,1}([\mathbf{h}_i;\mathbf{h}_t]))}. \tag{5.4}$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are vector representing the semantic meanings of nodes i, j respectively. $T_{2k,1}$ is an affine transformation from $\mathbb{R}^{2k}$ to $\mathbb{R}$. Equation 5.4 is essentially equivalent to

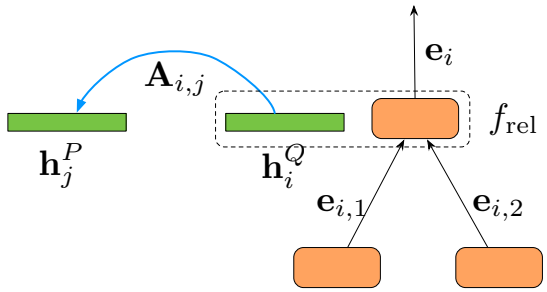


Figure 5.8: Network for entailment inference.

the widely used attention calculation in neural networks [3].

Once the alignment is known, we build another network to predict the final entailment label in a bottom-up manner. This recursive network consists of modules recursively applied at each tree node on the hypothesis tree. Figure 5.8 shows the recursive module.

In Figure 5.8, we use a vector $\mathbf{e}_i \in \mathbb{R}^r$ to represent the entailment relation at node $i \in Q$. We can induce the probabilities final entailment labels from the entailment vector $\mathbf{e}_{\text{root}}$ using a single non-linear layer with softmax.

$$\Pr(y|\mathbf{A}, P, Q) = \text{softmax}(\tanh(T_{r,3}(\mathbf{e}_{\text{root}}))).$$

At each hypothesis node i, $\mathbf{e}_i$ is calculated recursively given the meaning representations at the corresponding tree node $\mathbf{h}_i^Q$, the meaning representations of the aligned premise tree node $\mathbf{h}_j^P$, $j \in P$, and the entailment information from i's children $\mathbf{e}_{i,1}, \mathbf{e}_{i,2}$:

$$\mathbf{e}_i = f_{\text{rel}}([\mathbf{h}_i^Q; \sum_{j \in P} \mathbf{A}_{i,j}\mathbf{h}_j^P], \mathbf{e}_{i,1}, \mathbf{e}_{i,2}). \tag{5.5}$$

Finally we talk about how to compute the meaning representation $\mathbf{h}$ for each tree node in the premise tree and the hypothesis tree. In general, $\mathbf{h}_i$ should be calculated from the meaning representations $\mathbf{h}_{i,1}$, $\mathbf{h}_{i,2}$ of its two children if node i is an internal node, otherwise $\mathbf{h}_i$ should be calculated based on the word $\mathbf{x} \in \mathbb{R}^d$ in the leaf.

$$\mathbf{h}_i = f_{\text{MR}}(\mathbf{x}_i, \mathbf{h}_{i,1}, \mathbf{h}_{i,2}). \tag{5.6}$$

Note the similarity between Equation 5.5 and Equation 5.6: both of them are calculated recursively bottom-up from children nodes. This means we can use a similar composition function $f(\cdot)$ to calculate them.

We have various choices for composition function $f(\cdot)$. For example, we can use simple RNN functions [47]. Alternatively, we can use a convolutional layer to extract features from $\mathbf{x}_i, \mathbf{h}_{i,1}, \mathbf{h}_{i,2}$ (or $[\mathbf{h}_i^Q; \sum_{j \in P} \mathbf{A}_{i,j} \mathbf{h}_j^P], \mathbf{e}_{i,1}, \mathbf{e}_{i,2}$), and use pooling as aggregation to form $\mathbf{h}_i$ (or $\mathbf{e}_i$). In this paper we choose Tree-LSTM [50]. Our model is independent to this composition function and any high-quality composition function is sufficient for us to infer the meaning representations and entailment representations.

One thing worth mentioning is that in Tree-LSTM, besides the children representations $\mathbf{h}_{i,1}$, $\mathbf{h}_{i,2}$ (or $\mathbf{e}_{i,1}$, $\mathbf{e}_{i,2}$), the composition function also takes another pair of memory representations $\mathbf{c}_{i,1}$ and $\mathbf{c}_{i,2}$ as input. Take the calculation of the meaning representations $\mathbf{h}_i$ as an example. The Tree-LSTM composition function can be written as:

$$[\mathbf{h}_i; \mathbf{c}_i] = \text{LSTM}(\mathbf{x}_i, [\mathbf{h}_{i,1}; \mathbf{c}_{i,1}], [\mathbf{h}_{i,1}; \mathbf{c}_{i,2}]).$$

In practice, we use the above $\text{LSTM}(\cdot, \cdot, \cdot)$ function as f_{rel} and f_{MR}. But we only expose the output $\mathbf{h}_i$ (or $\mathbf{e}_i$) to the above layers and keep the memory $\mathbf{c}_i$ visible only to the LSTM function.

Following [63], we can summarize the LSTM function as following:

$$\begin{pmatrix} \mathbf{i}_i \\ \mathbf{f}_{i,1} \\ \mathbf{f}_{i,2} \\ \mathbf{o}_i \\ \mathbf{u}_i \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} T_{d+2k,k} \begin{pmatrix} \mathbf{x}_i \\ \mathbf{h}_{i,1} \\ \mathbf{h}_{i,2} \end{pmatrix}$$

$$\mathbf{c}_i = \mathbf{i}_i \odot \mathbf{u}_i + \mathbf{f}_{i,1} \odot \mathbf{c}_{i,1} + \mathbf{f}_{i,2} \odot \mathbf{c}_{i,2},$$
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i),$$

where $\mathbf{i}_i$, $\mathbf{f}_{i,1}$, $\mathbf{f}_{i,2}$, $\mathbf{o}_i$ represent the input gate, two forget gates for two children nodes, and the output gate respectively. $T_{d+2k,k}$ is an affine transformation from $\mathbb{R}^{d+2k}$ to $\mathbb{R}^k$.

5.3.1.3 Dual Alignment

We further improve our alignment in Equation 5.4, which does not consider any structural information of the current tree, nor any alignment information from the premise tree.

We can take a closer look at our conceptual example in Figure 5.6. Note that the alignment have, to some extent, a symmetric property: if a premise node j is most relevant to a hypothesis node i, then the hypothesis node i should also be most relevant to premise node j. For example, in Figure 5.6, the premise phrase "hugging one another" contradicts the hypothesis word "sleeping". In the perspective of the premise tree, the hypothesis word "sleeping" contradicts the known claim "hugging one another". This indicates that calculating the alignments from both sides can reduce some uncertainty. This technique is similar to the widely used forward and reversed alignment technique in the machine translation area.

The calculation of dual alignment is straightforward: we compute the alignment $\mathbf{A}$ from hypothesis to premise and also the alignment $\mathbf{A}^R$ from premise to hypothesis and then use the element-wise product

$$\mathbf{A}^* = \mathbf{A} \cdot \mathbf{A}^R$$

as the alignment to feed into the entailment composition module. This element-wise product is a mimic of the intersection operation over two alignments in machine translation.

5.3.2 Latent Variable Structured Learning with Neural Model

The training of the above model (Equation 5.3) is more involved when taking the latent structure, i.e., the alignment $\mathbf{A}$, into consideration. Here we first write down the stepwise

objective function to minimize during the training

$$\begin{aligned}\ell(y, P, Q) &= -\log \Pr(y|P, Q) \\ &= -\log \sum_{\mathbf{A}} \Pr(y, \mathbf{A}|P, Q) \\ &= -\log \sum_{\mathbf{A}} \Pr(\mathbf{A}|P, Q) \cdot \Pr(y|\mathbf{A}, P, Q) \\ &= -\log \mathbb{E}_{\Pr(\mathbf{A}|P,Q)}[\Pr(y|\mathbf{A}, P, Q)] \end{aligned} \tag{5.7}$$

Our model can also be viewed as a special case of latent variable CRF: instead of calculating the final probability based on a structured output, e.g., the coarse label sequence, we calculate the final probability based on a label y, but both the structured output and the label rely on the latent variable, e.g., refined label sequence in Section 4.1.1 or the alignment $\mathbf{A}$ in our case.

In general, our approach is similar to the training of a latent variable CRF model, i.e., minimizing the stepwise negative log likelihood. But this means we need to enumerate all possible alignments $\mathbf{A} \in \{0, 1\}^{|Q|\times|P|}$, the number of which grows exponentially fast with respect to $|P|$ and $|Q|$. This computation is extremely time-consuming and is difficult to handle for for latent variable CRF models.

Here we propose two approaches to approximately address this issue. The first one is based on sampling, and the second one is based on further approximation in the stepwise objective function 5.7.

5.3.2.1 Approximation via Sampling

Here instead of directly minimizing the stepwise objective Equation 5.7, we minimize an upper-bound of it.

Consider Equation 5.7. Since logarithm is a concave function, using Jensen's inequality, we have

$$\begin{aligned}\ell(y, P, Q) &= -\log \mathbb{E}_{\Pr(\mathbf{A}|P,Q)}[\Pr(y|\mathbf{A}, P, Q)] \\ &\leq -\mathbb{E}_{\Pr(\mathbf{A}|P,Q)} \log[\Pr(y|\mathbf{A}, P, Q)] = \bar{\ell}(y, P, Q).\end{aligned}$$

The upper-bound $\bar{\ell}(y, P, Q)$ is easier to minimize since we can approximate the ex-

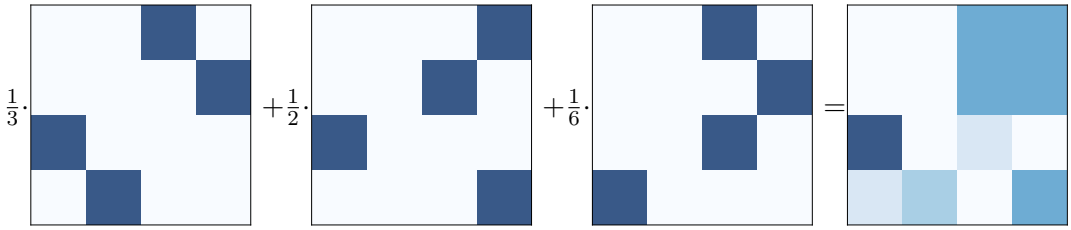


Figure 5.9: Expected alignment over 3 alignments with probability distribution of $(\frac{1}{3}, \frac{1}{2}, \frac{1}{6})$. Each alignment is a matrix of $\{0,1\}^{4\times4}$, and the expected alignment is a matrix of $\mathbb{R}^{4\times4}$.

pectation over the alignment $\mathbf{A}$ via sampling.

In detail, for every input sample (P, Q, y), we first randomly sample an alignment $\mathbf{A}$ based on distribution $\Pr(\mathbf{A}|P, Q)$, and then infer the final entailment probability with alignment $\mathbf{A}$ fixed.

5.3.2.2 Approximation in Objective Function

Alternatively, we can also optimize an approximated objective function. Notice that, as long as the calculation $\Pr(y|P, Q)$ only consists of linear calculations, simple nonlinearities like tanh and softmax, we can have following simplification via first-order Taylor approximation [60]:

$$\begin{aligned}\ell(y, P, Q) &= -\log \mathbb{E}_{\Pr(\mathbf{A}|P,Q)}[\Pr(y|\mathbf{A}, P, Q)] \\ &\approx -\log \Pr(y|\mathbb{E}_{\Pr(\mathbf{A}|P,Q)}[\mathbf{A}], P, Q) \\ &= -\log \Pr(y|\tilde{\mathbf{A}}, P, Q) = \tilde{\ell}(y, P, Q),\end{aligned}$$

which indicates that, instead of enumerating over all possible alignments and calculating the label probability for each alignment, we can use the label probability for the expected alignment as an approximation:

$$\tilde{\mathbf{A}} = \mathbb{E}_{\Pr(\mathbf{A}|P,Q)}[\mathbf{A}] \in \mathbb{R}^{|Q|\times|P|}.$$

An example of the expected alignment is shown in Figure 5.9.

This approximated objective $\tilde{\ell}(y, P, Q)$ is easier to minimize since the computation

of the expected alignment $\tilde{\mathbf{A}}$ can be decomposed into node-level:

$$\tilde{\mathbf{A}}_{i,j} = \Pr(\mathbf{A}_{i,j} = 1|P, Q).$$

The dual alignment technique can be applied similarly to the expected alignment calculation.

5.3.3 Empirical Evaluation

We evaluate the performances of our structured attention model and structured entailment model on the Stanford Natural Inference (SNLI) dataset [8], which contains $\sim$ 570k sentence pairs. We use the binarized trees in SNLI dataset in the experiments.

For word embedding parameters we use GloVe [42] for initialization. During the training we do not update the word embeddings for known words. For the rest layers, we initialize the parameters uniformly between -0.05 and 0.05. The lengths of both the Tree-LSTM meaning representation k, and the entailment relation vector r are set to 150.

We train our model with Adam with hyper-parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate 0.001, minibatch size 32, and dropout rate 0.2.

5.3.3.1 Quantitative Evaluation

We compare the performances of our models with various existing methods [8, 45, 50, 56, 11]. For those methods that do not provide an official implementation for our task, we use our own implementations. The results are summarized in Table 5.7.

As a baseline, we first use the binary Tree-LSTM to induce the meaning representations of the premise tree and the hypothesis tree, and then predict the entailment relation based on the meaning representations of the root nodes (Row 7 of Table 5.7), which already outperforms the naïve sequence LSTM model (Row 6 of Table 5.7). We further add the simple attention of [45] on top of the binary Tree-LSTM (Row 8 of Table 5.7), but only achieve limited improvement, since we observe severe vanishing gradient problem during training.

Composing the entailment relation from bottom up significantly improves the per-

Method	k	$\|\theta\|_M$	Train	Dev.	
1	LSTM sent. embedding [8]	100	221k	84.8	77.6
2	Sparse Features + Classifier [8]	-	-	99.7	78.2
3	LSTM + word-by-word attention [45]	100	252k	85.3	83.5
4	mLSTM [56]	300	1.9m	92.0	86.1
5	LSTM-network [11]	450	3.4m	88.5	86.3
6	LSTM sent. embedding (our implement. of [8])	100	241k	79.0	78.4
7	Binary Tree-LSTM (our implementation of [50])	100	211k	82.4	79.9
8	Binary Tree-LSTM + simple RNN w/ attention	150	220k	82.4	81.8
9	Binary Tree-LSTM + Sampling	150	0.9m	85.0	84.2
10	+ dual alignment	150	0.9m	85.8	85.4
11	Binary Tree-LSTM + Expected Alignment	150	0.9m	87.0	86.4
12	+ dual alignment	150	0.9m	87.7	**87.2**

Table 5.7: Comparison between our structured model with other existing methods. Column k specifies the length of the meaning representations. $|\theta|_M$ is the number of parameters without the word embeddings.

formance. When using the sampling strategy (Section 5.3.2.1, Row 9 of Table 5.7), we observe an improvement of ~ 2.4, and adding the dual alignment technique (Row 10 of Table 5.7) brings another ~ 1.2 improvement.

Binary Tree-LSTM with expected alignment strategy (Section 5.3.2.2, Rows 11 and 12 of Table 5.7) achieves the best performance. Comparing to the sampling strategy, we find expected alignment learns faster and converges to a higher accuracy, which might be because in sampling we are only optimizing an upper-bound of the true objective function.

5.3.3.2 Qualitative Evaluation

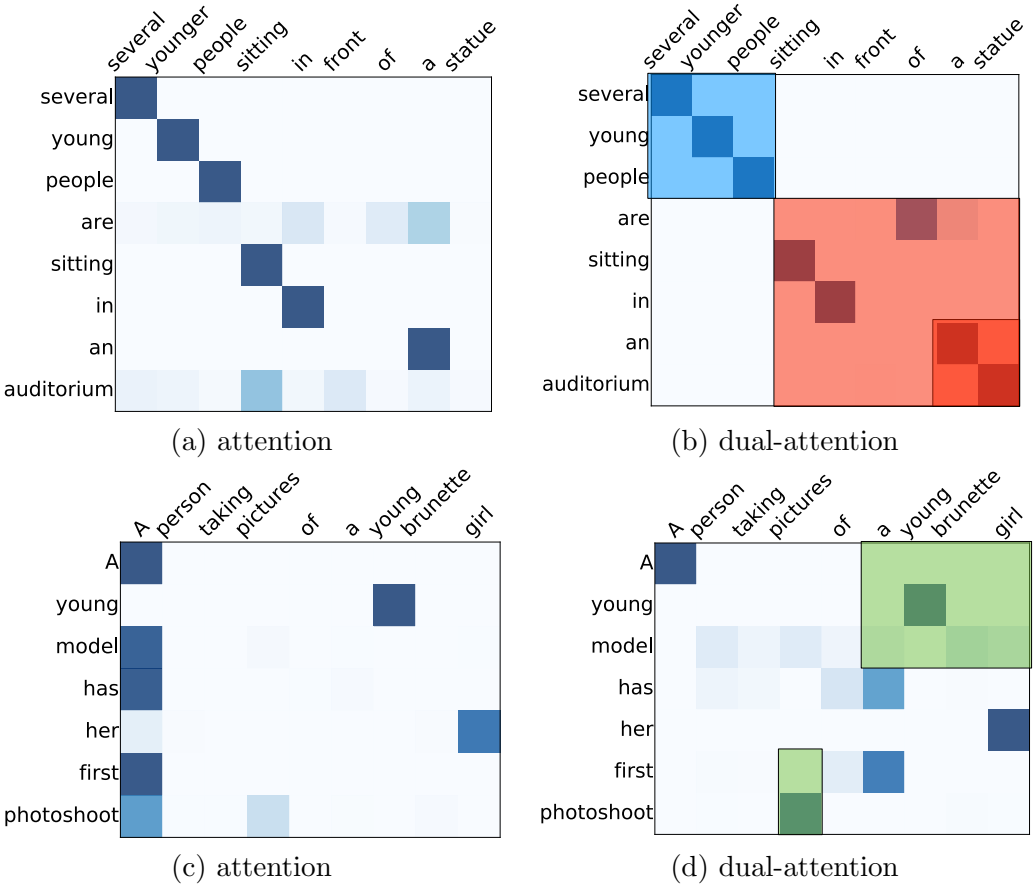
We further analyze the results of our model by showing two concrete examples taken from the SNLI dataset. Picking the most representative examples to demonstrate should be handled with great carefulness. Ideally we should randomly choose examples from SNLI. However, in practice we find that most correctly classified examples in the dataset are trivial, i.e., either with words insertion, deletion, or replacement, and many incorrectly classified examples involves common knowledge. It seems to be time-consuming to find meaning insights from randomly selected examples.

The two examples here are manually chosen with consideration of both generality and non-triviality: they both involve complex syntactic structure and multi-layers of entailment composition.

The first example is between the premise "several younger people sitting in front of a statue" and the hypothesis "several young people are sitting in an auditorium". Figure 5.10 (a) and (b) show the expected alignment and dual alignment. Without dual alignment, the premise word "sitting" tends to be mistakenly aligned to hypothesis word "auditorium", since in meaning representation level these words are more relevant semantically. But looking from the perspective of the premise tree corrects this mistake.

Figure 5.11 (a) draws the alignments of Figure 5.10 (b) between the tree nodes. It also shows how the final entailment relation is inferred from bottom up. Note that the hypothesis word "auditorium" contradicts the premise word "statue", which changes all the other entailment relations from entailment to contradiction.

The second example is even more interesting. The premise sentence is "a person taking pictures of a young brunette girl", and the hypothesis sentence is "a young model has her first photoshoot". First, in Figure 5.10 (c) and (d), we can observe that without dual alignment, many hypothesis words like "model", "first", and "photoshoot" are incorrectly aligned to premise word "a", which is similar to the garbage-collection phenomenon in machine translation. Dual alignment fixes all of these mistakes. Figure 5.11 (b) shows how the entailment is inferred internally. Note that in this example, there are two ambiguous spots. Firstly, "a young brunette girl" is not necessarily a "model". Secondly, "taking pictures" is not necessarily "her first photoshoot". Both of these two ambiguities are identified in Figure 5.11.



(a) attention (b) dual-attention

(c) attention (d) dual-attention

Figure 5.10: Attention matrices for exemplary sentence pairs. Note that, for brevity we *only show* the attentions between each word pair, and skip the attentions of tree nodes. Some important tree node alignments calculated by our model are highlighted using the colored boxes, where the colors of the boxes represent the entailment relations (see Figure 5.11). (a) (b) Premise: several younger people sitting in front of a statue. Hypothesis: several young people sitting in an auditorium. Dual-attention fixes the misaligned word "auditorium". (c) (d) Premise: A person taking pictures of a young brunette girl. Hypothesis: A young model has her first photoshoot. Dual-attention fixes the uncertain alignments for "photoshoot" and "model".

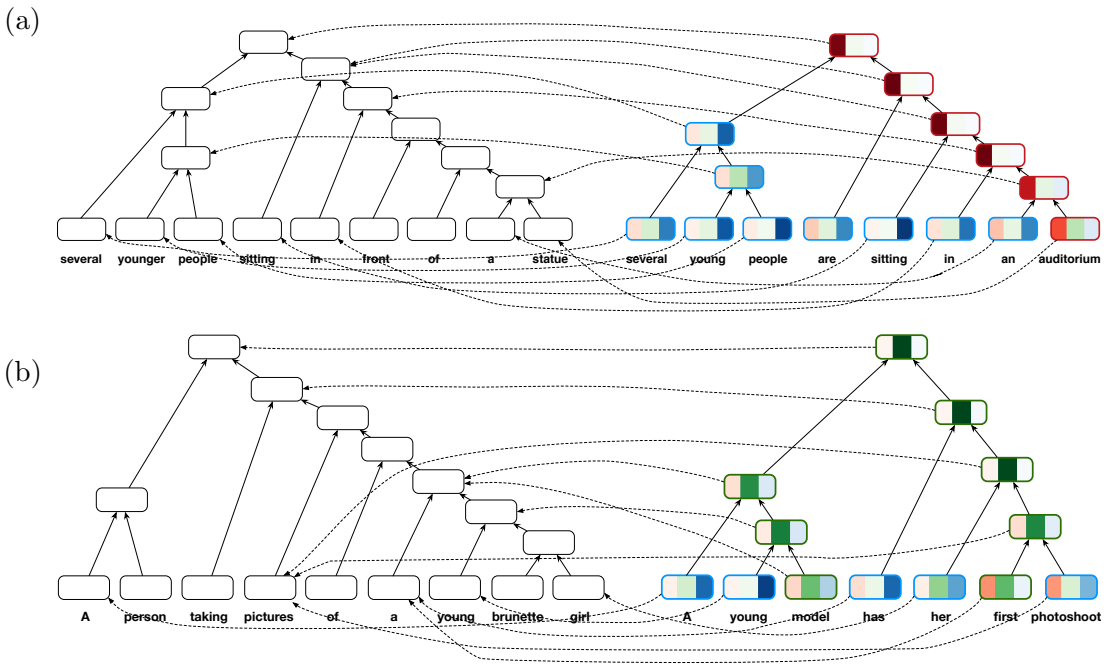


Figure 5.11: Examples illustrating entailment relation composition. (a) for Figure 5.10 (b); (b) for Figure 5.10 (d). For each hypothesis tree node, the dashed line shows to its most confident alignment. The three color stripes in each node indicate the confidences of the corresponding entailment relation estimation: red for contradiction, green for neutral, and blue for entailment. The colors of the node borders show the dominant estimation. Note: there is no strong alignment for hypothesis word "are" in (a).

Chapter 6: Summary

We reviewed some conventional structured learning algorithms for several popular NLP tasks. We discussed the theoretical properties of structured perceptron, CRF, and EM, and their applications in sequence labeling, constituent/dependency parsing, semantic parsing, machine translation and grapheme-to-phoneme conversion.

We can draw the relations between these structured learning methods in Table 6.1. At the very beginning we have the generative naïve bayes model, whose structured extension is HMM. By introducing latent variables like latent alignments in grapheme-to-phoneme conversion (Chapter 4.4) and rule extraction in machine translation to HMM we get EM model. On the other hand, if we model the system with conditional probability $\Pr(y|x)$ instead of joint probability $\Pr(x, y)$, naïve bayes model becomes logistic regression model (also called maximum entropy model), HMM model becomes CRF, and EM becomes latent variable CRF. This explains the similarity between the inference problems of EM and latent variable CRF. If we further approximate the inference problem of logistic regression with a Viterbi approximation, i.e., calculating the label with the maximal score instead of normalizing the probability, and train the logistic regression in an online fashion, we can get perceptron. Similarly CRF becomes structured perceptron, and latent variable CRF becomes latent variable structured perceptron. This relation hints that some theoretical properties of online latent variable CRF with inexact search can be applied to latent variable structured perceptron directly since latent variable structured

	binary/multi-class	+ structured learning	+ latent variables
generative	naïve bayes	HMM	EM
	+ conditional		
discriminative	logistic regression	CRF	latent variable CRF
	+ online + Viterbi (Chapter 3.2.3)		
	perceptron	structured perceptron	latent variable structured perceptron

Table 6.1: Relations between conventional structured and non-structured learning algorithms with and without latent variables.

perceptron can be viewed as a Viterbi version of online latent variable CRF.

We also demonstrated that latent variable structured learning as a flexible model to handle a variety of structured learning problems, especially for those problems where full annotation of the searching process is hard and usually only part of the reference derivation can be annotated. Our results in semantic parsing and machine translation show that latent variable structured learning algorithms can simplify the structured model. The results of latent variable structured learning outperform or on par with the conventional models. Our application in sentence entailment show that the latent variable structured learning technique is applicable to the neural models, and can effectively help the training.

Bibliography

[1] Gerry Altmann and Mark Steedman. Interaction with context during human sentence processing. *Cognition*, 30(3):191–238, 1988.

[2] Jimmy Ba, Ruslan R Salakhutdinov, Roger B Grosse, and Brendan J Frey. Learning wake-sleep recurrent attention models. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation (amr) 1.0 specification. In *In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. Seattle: ACL*, pages 1533–1544, 2012.

[5] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[6] Adam Berger. The improved iterative scaling algorithm: A gentle introduction. *Unpublished manuscript*, 1997.

[7] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, 2008.

[8] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[9] Victor Chahuneau, Noah Smith, and Chris Dyer. pycdec: A python interface to cdec. *The Prague Bulletin of Mathematical Linguistics*, 98:51–61, 2012.

[10] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics, 2005.

[11] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[12] David Chiang. Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228, 2007.

[13] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, 2002.

[14] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics, 2004.

[15] Koby Crammer, Michael Kearns, and Jennifer Wortman. Learning from multiple sources. *The Journal of Machine Learning Research*, 9:1757–1774, 2008.

[16] Harold Charles Daume. *Practical structured learning techniques for natural language processing*. ProQuest, 2006.

[17] Trinh Do, Thierry Arti, et al. Neural conditional random fields. In *International Conference on Artificial Intelligence and Statistics*, pages 177–184, 2010.

[18] Greg Durrett and Dan Klein. Neural crf parsing. In *Proceedings of the 43th annual meeting on association for computational linguistics*, 2015.

[19] Chris Dyer, Jonathan Weese, Hendra Setiawan, Adam Lopez, Ferhan Ture, Vladimir Eidelman, Juri Ganitkevitch, Phil Blunsom, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12. Association for Computational Linguistics, 2010.

[20] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[21] Mark Hopkins and Jonathan May. Tuning as ranking. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362. Association for Computational Linguistics, 2011.

[22] Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

[23] Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the*

Association for Computational Linguistics: Human Language Technologies, pages 142–151. Association for Computational Linguistics, 2012.

[24] Aravind K Joshi, Leon S Levy, and Masako Takahashi. Tree adjunct grammars. *Journal of computer and system sciences*, 10(1):136–163, 1975.

[25] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86, 2005.

[26] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[27] Shankar Kumar, Wolfgang Macherey, Chris Dyer, and Franz Och. Efficient minimum error rate training and minimum bayes-risk decoding for translation hypergraphs and lattices. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 163–171. Association for Computational Linguistics, 2009.

[28] Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. 2013.

[29] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1223–1233. Association for Computational Linguistics, 2010.

[30] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523. Association for Computational Linguistics, 2011.

[31] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[32] Percy Liang. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*, 2013.

[33] Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting*

of the Association for Computational Linguistics, pages 761–768. Association for Computational Linguistics, 2006.

[34] Bill MacCartney and Christopher D Manning. Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 521–528. Association for Computational Linguistics, 2008.

[35] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[36] André FT Martins, Noah A Smith, and Eric P Xing. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 342–350. Association for Computational Linguistics, 2009.

[37] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

[38] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014.

[39] Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer, 2003.

[40] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics, 2003.

[41] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[42] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543, 2014.

[43] Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 705–713. Association for Computational Linguistics, 2010.

[44] Slav Petrov and Dan Klein. Discriminative log-linear grammars with latent variables. In *Advances in Neural Information Processing Systems*, pages 1153–1160, 2007.

[45] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

[46] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[47] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

[48] Xu Sun. Towards shockingly easy structured classification: A search-based probabilistic online learning framework. *arXiv preprint arXiv:1503.08381*, 2015.

[49] Xu Sun, Takuya Matsuzaki, Daisuke Okanohara, and Jun'ichi Tsujii. Latent variable perceptron algorithm for structured classification. In *IJCAI*, volume 9, pages 1236–1242, 2009.

[50] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. 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 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics.

[51] Lappoon R Tang and Raymond J Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Machine Learning: ECML 2001*, pages 466–477. Springer, 2001.

[52] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems*, page None, 2003.

[53] Ran Tian, Yusuke Miyao, and Takuya Matsuzaki. Logical inference on dependency-based compositional semantics. In *Proceedings of ACL*, pages 79–89, 2014.

[54] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005.

[55] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[56] Shuohang Wang and Jing Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.

[57] Yotaro Watanabe, Junta Mizuno, Eric Nichols, Naoaki Okazaki, and Kentaro Inui. A latent discriminative model for compositional entailment relation recognition using natural logic. In *COLING*, pages 2805–2820, 2012.

[58] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*, 2015.

[59] Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational linguistics*, 23(3):377–403, 1997.

[60] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[61] Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(02):207–238, 2005.

[62] Heng Yu, Liang Huang, Haitao Mi, and Kai Zhao. Max-violation perceptron and forced decoding for scalable mt training. In *EMNLP*, pages 1112–1123, 2013.

[63] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[64] Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *UAI*, 2005.

[65] Luke S Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *EMNLP-CoNLL*, pages 678–687, 2007.

[66] Hao Zhang, Liang Huang, Kai Zhao, and Ryan McDonald. Online learning for inexact hypergraph search. In *Proceedings of EMNLP*, 2013.

[67] Kai Zhao and Liang Huang. Type-driven incremental semantic parsing with polymorphism. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2015.

[68] Kai Zhao, Liang Huang, and Mingbo Ma. Textual entailment with structured attentions and composition. *Proceedings of COLING 2016*, 2016.

[69] Kai Zhao, Liang Huang, Haitao Mi, and Abe Ittycheriah. Hierarchical mt training using max-violation perceptron. 2014.

[70] Hao Zhou, Yue Zhang, 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*, pages 1213–1222, 2015.