DISCRIMINATIVE TRAINING AND VARIATIONAL DECODING IN MACHINE TRANSLATION VIA NOVEL ALGORITHMS FOR WEIGHTED HYPERGRAPHS

by

Zhifei Li

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

April, 2010

Abstract

A hypergraph or "packed forest" is a compact data structure that uses structure-sharing to represent exponentially many trees in polynomial space. A *probabilistic/weighted* hypergraph also defines a probability (or other weight) for each tree, and can be used to represent the hypothesis space considered (for a given input) by a monolingual parser or a tree-based translation system (e.g., tree to string, string to tree, tree to tree, or string to string with latent tree structures).

Given a weighted/probabilistic hypergraph, we might ask three questions. What atomic operations can we perform on the weighted hypergraph? How do we set the weights in the hypergraph? Which particular translation (among the possible translations encoded in a hypergraph) should we present to an end user? These correspond to three fundamental problems: inference, training, and decoding, for which this dissertation will present novel techniques.

The atomic inference operations we may want to perform include finding one-best, k-best, or expectations over the hypergraph. To perform each operation, we may implement a dedicated dynamic programming algorithm. However, a more general framework to specify these algorithms is semiring-weighted logic programming. Within this framework, we first extend the expectation semiring, which is originally proposed for a finite state automaton, to a hypergraph. We then propose a novel *second-order* expectation semiring. These semirings can be used to compute a large number of expectations (e.g., entropy and its gradient) over the exponentially many trees presented in a hypergraph.

The weights used in a hypergraph are usually learnt by a discriminative training method. One common drawback of such method is that it relies on the existence of high-quality supervised data (i.e., bilingual data), which may be expensive to obtain. We present two unsupervised discriminative training methods: *minimum imputed-risk training*, and *contrastive language model estimation*, both can exploit monolingual English data to perform discriminative training. In minimum imputed-risk training, we first use a reverse translation model to impute the missing inputs, and then train a discriminative forward model by minimizing the expected loss of the forward translations of the missing inputs.

In contrast, the contrastive language model estimation does not use a reverse system. It first extracts a *confusion grammar*, then generates many alternative sentences (i.e., a contrastive set) for each English sentence using the confusion grammar, and finally trains a discriminative language model on the contrastive sets such that the model will prefer the

original English sentences (over the sentences in the contrastive sets).

During decoding, we are interested in finding a translation that has a maximum posterior probability (i.e., MAP decoding). However, this is intractable due to *spurious ambiguity*, a situation where the probability of a translation string is split among many distinct derivations (e.g., trees or segmentations). Therefore, most systems use a simple Viterbi decoding that approximates the string probability with its most probable derivation's probability. Instead, we develop a variational approximation, which considers all the derivations but still allows tractable decoding. Our particular variational distributions are parameterized as n-gram models. We also analytically show that interpolating these n-gram models for different n is similar to lattice-based minimum-risk decoding for BLEU. Experiments show that our approach improves the state of the art.

All the above methods have been implemented in an open-source machine translation toolkit **Joshua**. In this dissertation, the methods have mainly been applied to a machine translation task, but we expect that they will also find applications in other areas of natural language processing (e.g., parsing and speech recognition).

Primary Advisor: Sanjeev Khudanpur
Secondary Advisor: Jason Eisner
Readers: Sanjeev Khudanpur, Jason Eisner
Committee: Sanjeev Khudanpur, Jason Eisner, Chris Callison-Burch

Acknowledgements

First, I would like to thank my advisers, Sanjeev Khudanpur and Jason Eisner, for many helpful and insightful technical discussions during the course of my graduate study. They have served as two oracles to me, in many aspects. Sanjeev knows beforehand every math formula I may need to use in my research, and Jason knows in advance every algorithm I may want to implement. They are also great teachers, knowing how to explain very sophisticated material in simple terms. I thank Sanjeev for bringing me into the wonderful world of statistical natural language processing, which I may never have ventured into, had I not taken his intensive class about statistical speech recognition. I thank Jason for shaping me into a better presenter and for changing how I approach machine translation as a formal problem.

I would also like to thank Chris Callison-Burch, who essentially has acted as my third advisor, in many aspects. He is always supportive of me and eager to promote my work to folks outside JHU. He has also provided many valuable insights and suggestions about my work. Since his arrival at Johns Hopkins University, the research on MT at CLSP has become more involved, which has benefitted me a lot.

I am very grateful to David Yarowsky, with whom I collaborated for two papers. He is always so gentle, and is able to identify the key intuitions that make our ideas work.

My work has also greatly benefited from regular discussions with Markus Dreyer and Jason Smith. Special thanks go to Ziyuan Wang, who helped me to run the experiments in chapters 4 and 5.

I am also very grateful for many discussions with Professor Damianos Karakos, and many students at CLSP such as Jia Cui, Yonggang Deng, Anoop Deoras, Nikesh Garera, Arnab Ghoshal, Delip Rao, Ariya Rastrow, David Smith, Yi Su, Roy Tromble, Balakrishnan Varadarajan, Chris White, Puyang Xu, Omar Zaidan, and Haolang Zhou.

I also want to thank Philip Resnik at UMD for allowing me to attend his group's MT meeting in 2007, which gives me the opportunity to interact with his great students. A special thank goes to Chris Dyer, whom I still regularly interact with.

I also want to thank all the members in the Joshua project for a great team experience. They are: Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese and Omar Zaidan. I also want to thank Desiree Cleves for proofreading my dissertation.

I also want to thank my mentor at MSR, Patrick Nguyen, for hosting me as a summer

intern in 2007. I have also benefitted from discussions with many non-JHU Chinese fellows when attending conferences. They are Jianfeng Gao, Xiaodong He, Liang Huang, Mu Li, Qun Liu, Yang Liu, Bing Zhao, Hao Zhang, and Joy Zhang.

Finally, I want to thank my wife, Zhuangxiang He, for her unconditional love and her sacrifice for me during the many years we have lived together. I also thank my other family members in China for their love and support.

Zhifei Li, April 2010

Contents

List of Figures

List of Tables

Chapter 1

Summary of the Dissertation

In this chapter, we first review some basic background of statistical machine translation,[1] and then describe the contributions and roadmap of the dissertation. In the discussion, we will mainly use a translation task from Chinese to English as an example, although the methods are mostly language interdependent.

1.1 Statistical Machine Translation Pipeline

In a statistical machine translation (SMT) task (e.g., from Chinese to English), we are often given some *bilingual training data* (e.g., Chinese sentences and their corresponding English references), *monolingual training data* (e.g., English text only), and some *test data* (i.e., Chinese text unseen in the training data). Our goal is to first build a translation system on the training data, and then use the system to generate good translations for the test data. The pipeline of a typical SMT system is shown in Figure 1.1, where the upper and lower parts correspond to training and testing, respectively. There are four major components in the pipeline and we will describe them below.

1.1.1 Training Translation Models on Bilingual Data

Given the bilingual training data, we train translation model(s) on it and the training usually involves several steps: *data normalization*, *word alignment*, *grammar extraction*, and *parameter estimation*. Figure 1.2 shows an instance of the steps where we are given a toy bilingual training corpus containing only one sentence pair.

The data normalization step converts the bilingual data to a canonical form. For example, we may convert all the English words to lower case, and separate punctuation from the words. For a language (e.g., Chinese) where the words are not space-separated in the raw text, we often segment a sentence into words by adding spaces. These normalization

[1] For a more comprehensive introduction, please refer to Koehn (2010).

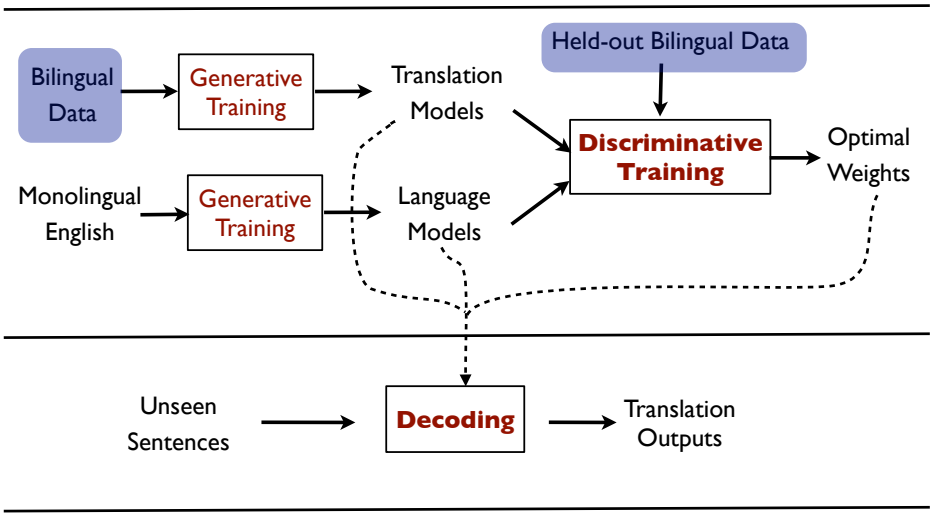


Figure 1.1: **A typical pipeline for a statistical machine translation system.** The upper and lower parts correspond to training and testing, respectively. This dissertation mainly focus on the *discriminative training* and *decoding* components. Note that the bilingual data used for discriminative training usually has a much smaller size (but much better quality) than the bilingual data used for generative training of the translation models.

operations are usually language dependent. Figure 1.2(a) shows the raw bilingual data and Figure 1.2(b) shows the version after performing normalization.

The bilingual corpora provided are usually sentence-aligned,[2] but seldom contain word-alignments, which correspond the words/phrases in the two languages to each other. Therefore, we employ a word-alignment algorithm (Brown, Pietra, Pietra, and Mercer, 1993) to discover such correspondence. Figure 1.2(c) shows the word alignment for the illustrative one sentence corpus.

With the word-aligned corpora, we can then extract a translation grammar. The grammar can be word-based (Brown et al., 1993), phrase-based (Koehn, Och, and Marcu, 2003), hierarchical phrase-based (Chiang, 2005, 2007), or syntax-based (e.g., Galley, Graehl, Knight, Marcu, DeNeefe, Wang, and Thayer, 2006). In this dissertation, we use a hierarchical phrase-based machine translation system, called Hiero (Chiang, 2007) (see more details below). However, our methods are quite general and can also be applied to the other kinds of MT systems just listed. The first column of Figure 1.2(d) shows the possible Hiero translation rules we extract from the training sentence pair.

Each rule in the grammar will be associated with a probability/weight that is estimated from the training data. One usually uses *maximum likelihood estimation* (or *generative training*) where the probability of a rule is simply the relative frequency of the rule (i.e., the number of times the given rule got extracted divided by the total number of times any

[2]They might be only document-aligned, in which case we also need to discover the sentence alignments.

垫子上的猫
dianzishangdemao

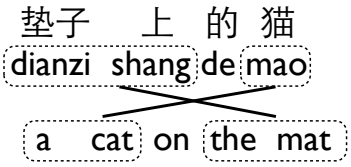
A cat on the mat

(a) raw bi-text

垫子 上 的 猫
dianzi shang de mao

a cat on the mat

(b) bi-text after normalization



(c) bi-text after word alignment

Rule	**Probability**
$X \rightarrow \langle$ dianzi shang , the mat $\rangle$	0.2
$X \rightarrow \langle$ mao , a cat $\rangle$	0.2
$X \rightarrow \langle$ dianzi shang de X_0 , X_0 on the mat $\rangle$	0.2
$X \rightarrow \langle X_0$ de mao , a cat on $X_0 \rangle$	0.2
$X \rightarrow \langle X_0$ de X_1 , X_1 on $X_0 \rangle$	0.2

(d) Hiero rules extracted from the aligned bi-text. Each rule is associated with a probability that is estimated from the training data. In our example, we use maximum likelihood estimation and thus all the probabilities are 0.2 ($=\frac{1}{5}$) since we have only five rules and each of them has been seen once in the simple training corpus. In practice, we may have thousands of training sentence pairs, from which many more rules can be extracted.

Figure 1.2: **A simple example to show the steps in extracting a translation model.** The bi-text has only one sentence pair. For the Chinese side of the bi-text, we show both the Chinese characters and its *Pinyin* (for the convenience of non-Chinese speakers). We show only the *Pinyin* in the rules in Table (d).

rule got extracted). In practice, one usually smoothes the probability distribution. The second column of Figure 1.2(d) shows the rule probabilities before smoothing.

Hierarchical Phrase-based Machine Translation

In Hiero, a synchronous context-free grammar (SCFG) is extracted from automatically word-aligned corpora. An SCFG comprises a set of source-language terminal symbols T_S, a set of target-language terminal symbols T_T, a shared set of nonterminal symbols N, and a set of rules of the form

$$X \rightarrow \langle \gamma, \alpha, \sim \rangle \,, \tag{1.1}$$

where $X \in N$, $\gamma \in [N \cup T_S]^*$ is a (mixed) sequence of nonterminals and source terminals, $\alpha \in [N \cup T_T]^*$ is a (mixed) sequence of nonterminals and target terminals, and $\sim$ is a one-to-one correspondence or *alignment* between the nonterminal elements of γ and α.

Two example Hiero rules for Chinese-to-English translation are

$$X \rightarrow \langle \text{ mao , a cat } \rangle$$

$$X \rightarrow \langle X_0 \text{ de } X_1 , X_1 \text{ of } X_0 \rangle$$

In the rules above, the alignment between nonterminals, i.e $\sim$, is implicitly encoded by the subscripts on the nonterminals. In Hiero, there is only a single nonterminal X, while a syntax-based grammar (Galley et al., 2006) contains a richer set of linguistically informed nonterminals like noun phrase (NP) and verb phrase (VP).[3] The first rule shows that we can translate the Chinese word *mao* to English words *a cat*.[4] The second rule shows that the two phrases (represented by X_0 and X_1) around *de* in Chinese will get reordered around *of* in the English. Such reordering is very typical between Chinese and English.

Hiero also includes the following two *glue* rules such that each input string will have at least one derivation tree that is allowed by the grammar.

$$S \rightarrow \langle X_0 , X_0 \rangle$$

$$S \rightarrow \langle S_0 X_1 , S_0 X_1 \rangle$$

where S is the *goal* symbol.

As mentioned above, a rule in Hiero may contain both terminal (e.g., the word *of*) and nonterminal (e.g., X_0) symbols. In a given rule, the number of nonterminals in the Chinese is the same as that in the English (this is what the word *synchronous* implies). The **arity** of a rule is defined as the number of nonterminals in the rule. For example, the arities for the two rules shown above are zero and two, respectively, while the arity for the third and forth rule in Figure 1.2(d) is one.

1.1.2 Training Language Models on Monolingual Data

The translation model tells us what kinds of English translations we could generate given a Chinese input. Intuitively, we have a prior belief about what should be a good English sentence (e.g., whether it is fluent). Therefore, to certain extent, we should be able to judge the goodness of a translation output, even without looking at the source-language input. For example, we have a prior belief that "*a cat on the mat*" should be more likely than "*a cats on the mat*" since the bad bigram "*a cats*" is much less likely than the natural one "*a cat*." This intuition is incorporated into the SMT system through using a language model, which assigns a probability to an English string. We usually use a so-called n-gram

[3] To be precise, Hiero also contains a nonterminal S, which is the goal symbol.

[4] More precisely, *mao* is the *Pinyin* of the Chinese word 猫. *Pinyin* is the most commonly used romanization system for Chinese, and we use it for the convenience of non-Chinese speakers.

model, under which the probability of a sentence is the product of the probabilities of the n-grams occurring in the sentence. The n-gram probabilities are trained on the monolingual English data by using maximum likelihood estimation (i.e., the probability of an n-gram is its relative frequency in the training corpus), often with some smoothing.

1.1.3 Discriminative Training of Relative Weights Among Models

With the translation and language models, how much should we trust each when we use them to score translation outputs for test data? Intuitively, we can assign a weight to each model, and trust the model proportionally to its weight. These weights are usually found through a training algorithm such as minimum error rate training (MERT) (e.g., Och, 2003). Note that a translation model itself may have many translation rules and each rule has a weight (or probability) (see Figure 1.2(d)). Similarly, an n-gram language model may have many n-grams and each has a weight. These weights (for individual rules or n-grams) are different from the relative weights among different models. Usually, the weights inside a translation/language model are trained in a generative way as mentioned before, while the relative weights among models are trained in a discriminative manner, which directly optimizes the translation performance (see Chapter 2 for more details).

1.1.4 Decoding for Test Data

Using the bilingual and monolingual training data, we have trained an SMT system (by following the pipeline in Figure 1.1), which has translation and language models, and the relative weights among the models. Now, we can generate translation outputs for unseen test data by using the trained system. For example, given a test Chinese sentence "*dianzi shang de gou*" (whose Chinese characters are "垫子 上 的 狗"), we may generate a translation output "*the dog on the mat*" assuming that the test translation grammar contains a rule

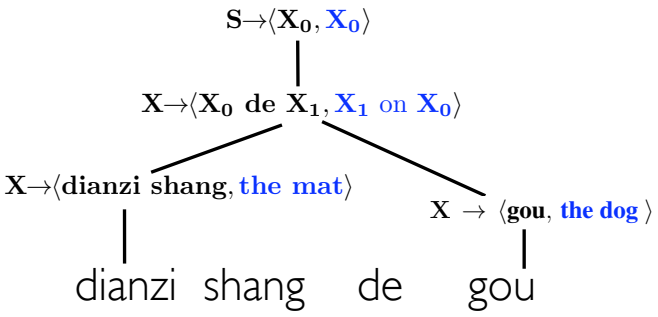
$$X \rightarrow \langle \text{ gou, the dog } \rangle,$$

in addition to those rules extracted from the training example in Figure 1.2(c).

Figure 1.3 shows the test grammar and the translation output. Note that the system generates a *derivation tree*, instead of just the translation *string* itself. In general, a derivation tree is composed of a sequence of rules, and it contains both an input sentence and an output sentence, which are called its *yields*. As an example, the tree shown in Figure 1.3(b) is composed of the rules in the test grammar shown by Figure 1.3(a), and its Chinese and English yields are "*dianzi shang de gou*" and "*the dog on the mat*", respectively. Of course, the system may consider many other candidate translations due to ambiguities in language translation (see Section 1.2).

X	$\rightarrow$	$\langle$ dianzi shang , the mat $\rangle$
X	$\rightarrow$	$\langle$ gou , the dog $\rangle$
X	$\rightarrow$	$\langle X_0$ de X_1 , X_1 on $X_0 \rangle$
S	$\rightarrow$	$\langle X_0$, $X_0 \rangle$

(a) Hiero rules in the test grammar



(b) derivation tree whose English yield is "the dog on the mat".

Figure 1.3: **Example of decoding a test sentence.** For the test sentence "dianzi shang de gou", the system uses the test grammar of (a) to generate a derivation tree in (b)

1.1.5 Focus of this Dissertation

As we have already seen, the machine translation pipeline involves in many components. In this dissertation, we will mainly focus on the discriminative training (of Section 1.1.3) and the decoding (of Section 1.1.4). As mentioned in Section 1.1.3, traditionally, the only weights that get discriminatively trained are those relative weights among different models. The number of such weights is usually very small (e.g., less than ten). In this dissertation, we aim to tune a large number of weights discriminatively. For example, we may want to discriminatively tune a weight for each individual translation rule in a translation model and a weight for each individual n-gram in a language model.

1.2 Ambiguity in Language Translation

As in many other natural language processing (NLP) problems, ambiguity is a central issue in machine translation. Broadly speaking, there are two kinds of ambiguities in translation: *translation-sense ambiguity* and *spurious ambiguity*.

1.2.1 Translation-Sense Ambiguity

In natural language, the same word may have different senses/meanings, depending on the context. For example, the word "bank" can refer to either a financial bank or the edge of a river. This kind of ambiguity is called *word sense ambiguity*. Clearly, when we translate such an ambiguous word, different translations should be used depending on the context. For example, a translation model (for a translation task from English to Chinese) may contain two rules as follows,

$$X \rightarrow \langle \text{ bank , he an} \rangle$$

$$X \rightarrow \langle \text{ bank , yin hang} \rangle$$

where they have the same English side (i.e., *bank*), but have different Chinese sides, which have different meaning in Chinese. Specifically, the first rule has the Chinese *he an*, which means the edge of a river, while the second one has the Chinese *yin hang*, which means a financial bank. We call such ambiguity *translation-sense ambiguity*. The example above involves rules with arity zero. In Hiero, it is equally possible to have rules with arity one or higher that have the same source-language side but different target-language sides. Figure 1.4 shows how different translation patterns (for the same source-side "X_0 de X_1") can be extracted from different training examples. These rules involve using different prepositions (*of* or *on*) or different ordering (with or without reordering). This indicates why the machine translation task is difficult.

1.2.2 Spurious Ambiguity

Intuitively, given an input sentence, the main job of an SMT system should be to disambiguate the input sentence and produce an appropriate translation string by carefully choosing translation senses. However, typical MT systems (Koehn et al., 2003; Chiang, 2007) will also recover a particular derivation of the output string, which specifies a tree or segmentation and its alignment to the input string. The competing derivations of a string are interchangeable for a user who is only interested in the string itself, so a system that unnecessarily tries to choose among them is said to be resolving *spurious ambiguity*. To emphasize, while translation-sense ambiguity will lead to different translation strings, spurious ambiguity does not.

Figure 1.5 shows two examples of spurious ambiguity. In particular, Figure 1.5(a) shows an example of spurious *segmentation* ambiguity occurring in regular phrase-based systems (e.g., Koehn et al., 2003), where different segmentations lead to the same translation string. In contrast, Figure 1.5(b) shows an example of spurious *tree* ambiguity that occurs in Hiero (Chiang, 2007), where different derivation trees yield the same string.

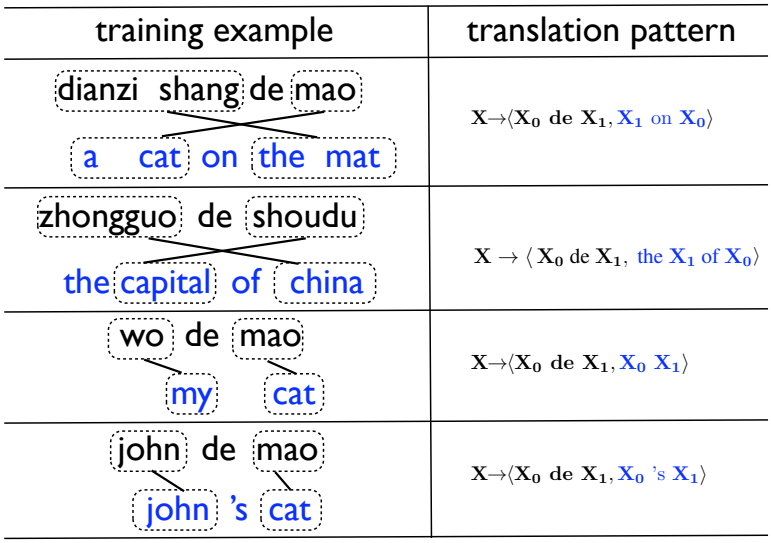


Figure 1.4: From different training examples, we can extract different translation patterns for the same Chinese side (i.e., X_0 de X_1). The first two patterns involve in reordering between X_0 and X_1, while the last two do not.

1.3 Hypergraph to Represent Ambiguity

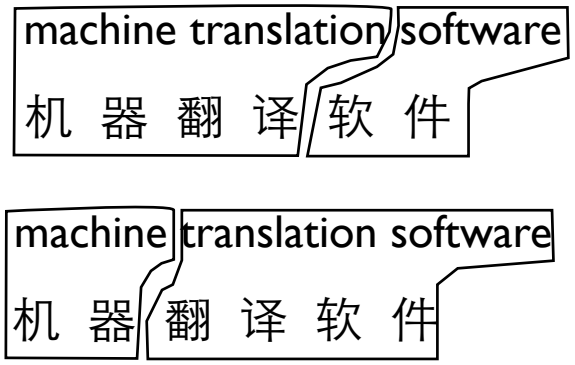
Given an input sentence, an SMT system may consider a huge number of possible candidate translations (along with the derivations), due to the two kinds of ambiguities discussed above. Therefore, brute-force ways of storing and operating on the candidates are not feasible, and we usually use a data structure called a hypergraph (Gallo, Longo, Pallottino, and Nguyen, 1993).

Informally, a hypergraph is a compact data structure that can store exponentially many different translation strings (due to translation-sense ambiguity) where each string in turn may correspond to many distinct derivation trees (due to spurious ambiguity). Figure 1.6 shows a simple hypergraph (Figure 1.6(a)), which encodes four different derivation trees (figures 1.6(b)–1.6(e)) generated for the input Chinese sentence "*dianzi shang de mao*". Note that this hypergraph encodes only translation-sense ambiguity, though in general it can additionally encode spurious ambiguity.

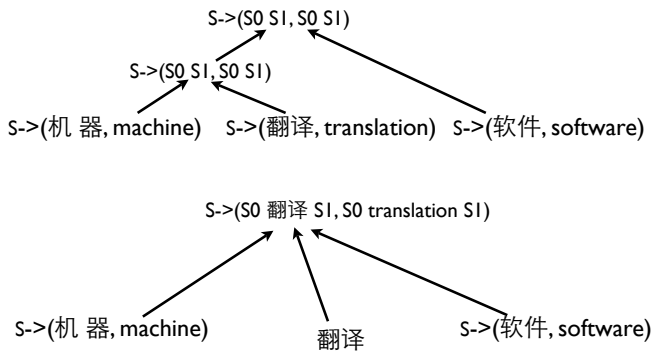
Formally, a hypergraph is a pair $\langle V, E \rangle$, where V is a set of *nodes* (vertices) and E is a set of *hyperedges*, with each hyperedge connecting a *sequence* of *antecedent nodes* to a single *consequent node*.[5]

In our application, a node (alternatively called an *item*) is identified by the non-terminal symbol and the source span. A hyperedge represents an SCFG rule that has been "instanti-

[5]Strictly speaking, making each hyperedge designate a single consequent defines a *B-hypergraph* (Gallo et al., 1993).



(a) Segmentation ambiguity in phrase-based MT: two different segmentations lead to the same translation string.



(b) Tree ambiguity in syntax-based MT: two different derivation trees yield the same translation string.

Figure 1.5: Examples of spurious ambiguity

ated" at a particular position, and has a pointer to an antecedent item for each non-terminal symbol in the rule. We have a *goal node* at the root of a hypergraph which corresponds to the goal symbol S. In general, a node has one or more incoming hyperedges, which represent different ways of deriving the node. A node may be shared by many hyperedges. For example, in Figure 1.6(a), the node $\boxed{X \mid 0, 2}$ (and the node $\boxed{X \mid 3, 4}$ as well) is shared by four hyperedges that point to it. Also, a hyperedge may be shared by many different derivations. For example, in Figure 1.6(a), the hyperedge labeled with the rule $S \rightarrow \langle X_0, X_0 \rangle$ is shared by all the four derivation trees. By exploiting such sharing, a hypergraph can compactly represent exponentially many trees.

Hypergraphs are closely related to other formalisms like packed forests, AND/OR graphs, context-free grammars, and deductive systems (Shieber, Schabes, and Pereira, 1994; Nederhof, 2003). Also, any finite-state automaton (FSA)[6] can be encoded as a hyper-

[6] Informally, a finite state automaton is a graph composed of a finite number of states (or nodes) and

graph (in which every hyperedge is an ordinary edge that connects a *single* antecedent to a consequent). Specific to MT, a hypergraph can represent the hypothesis space of a broad range of tree-based translation system, e.g., tree-to-string (Quirk, Menezes, and Cherry, 2005; Liu, Liu, and Lin, 2006), string-to-tree (Galley et al., 2006), tree-to-tree (Eisner, 2003), or string-to-string with latent tree structures (Chiang, 2007). Therefore, the methods developed in this dissertation will have very broad applications.

1.3.1 Weighted Hypergraphs

The hypergraph shown in Figure 1.6(a) is un-weighted. We can obtain a weighted hypergraph by assigning a weight/score to each derivation d in the hypergraph as follows,

$$\text{score}(d) = f(d) \cdot \theta = \sum_j f_j(d)\theta_j \,, \tag{1.2}$$

where $f(d)$ is a feature vector, θ is a weight vector, and j indexes the feature dimensions. A feature can be any property of the derivation (and its Chinese and English yield). For example, we may have a feature for each translation rule (or each English bigram) in the derivation.

1.3.2 Probabilistic Hypergraphs

Going even further, we can obtain a probabilistic hypergraph by converting the scores to probabilities as follows,

$$p_\theta(d \mid x) = \frac{1}{Z_\theta(x)} e^{\gamma \cdot \text{score}(d)} \,, \tag{1.3}$$

where x is the input sentence, d is a derivation, γ is a scaling factor to adjust the sharpness of the distribution (the larger γ is, the more peaked the distribution is), and $Z_\theta(x)$ is a normalization constant defined as,

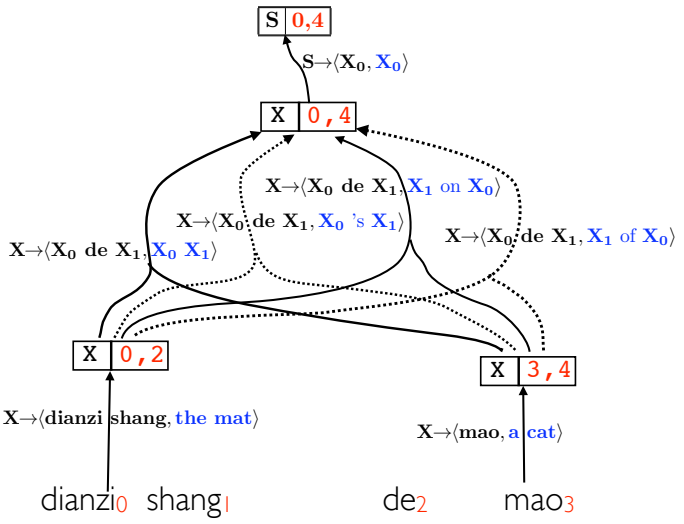
$$Z_\theta(x) = \sum_d e^{\gamma \cdot \text{score}(d)} \,. \tag{1.4}$$

By marginalizing over the latent variable d, we can obtain

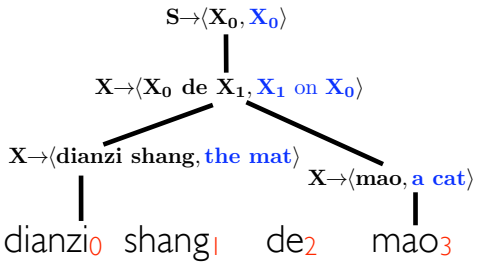
$$p_\theta(y \mid x) = \sum_{d \in \mathrm{D}(x,y)} p_\theta(d \mid x) \,, \tag{1.5}$$

where $\mathrm{D}(x, y)$ represents the set of derivation trees that yield the input string x and the output string y. In this way, **a weighted hypergraph encodes a probability distribution over derivation trees (and over translation strings as well)**.

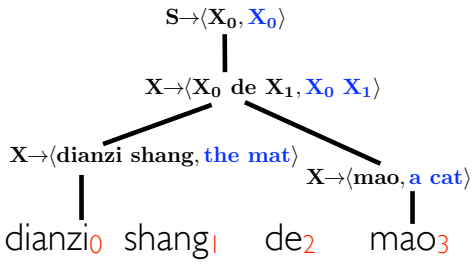
transitions (or edges) between those states. By assigning a weight to the state/transition, we can obtain a weighted finite state automaton (WFSA). In our presentation, we will simply use FSA regardless of whether it is weighted or not.



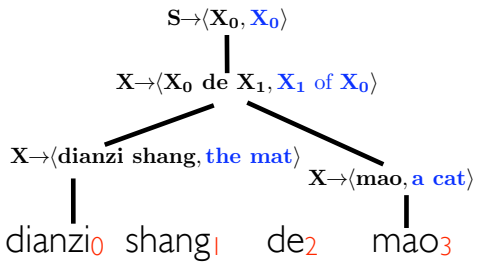
(a) This hypergraph encodes four different derivation trees as shown in the four figures below. Rectangles represent items (or nodes), where each item is identified by the non-terminal symbol and source span. An item has one or more incoming hyperedges, which represent different ways of deriving the item. A hyperedge consists of a rule, and a pointer to an antecedent item for each non-terminal symbol in the rule.



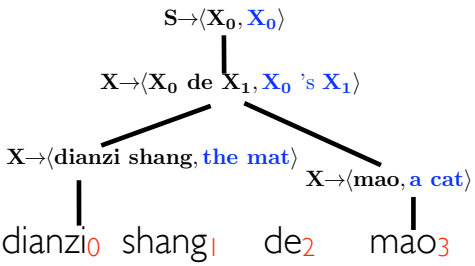
(b) Translation: **a cat on the mat**



(c) Translation: **the mat a cat**



(d) Translation: **a cat of the mat**



(e) Translation: **the mat 's a cat**

Figure 1.6: A toy hypergraph generated for the Chinese input "*dianzi shang de mao*," and the four derivation trees the hypergraph encodes.

1.4 Our Contributions

Given a weighted/probabilistic hypergraph, we might ask three questions. What atomic operations can we perform on the weighted hypergraph? How do we set the parameter vector θ that weights the derivations in the hypergraph? Which particular translation (among the possible translations encoded in a hypergraph) should we present to an end user? These correspond to three fundamental problems: inference, training, and decoding, which are the focus of the dissertation. Figure 1.7 shows the relationship between the three problems. The atomic inference operations form a basis for the more sophisticated tasks (e.g., training and decoding), and they can be exact or approximate.

Below, for each of the three fundamental problems, we will first give a brief review (see Chapter 2 for a more thorough review), and then present our contributions.

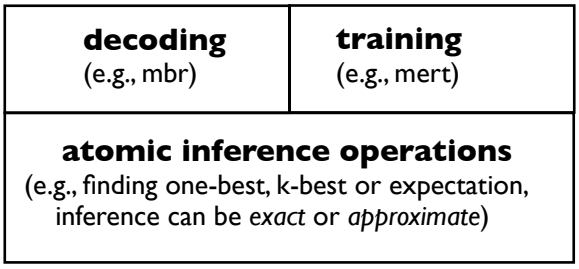


Figure 1.7: **Relations among inference, decoding and training.** Inference forms the basis from which more sophisticated tasks like decoding and training can be performed.

1.4.1 First- and Second-order Expectation Semirings

Inference: Previous Work

Given a hypergraph, we may want to perform many atomic inference operations over it. For example, we may want to find the most probable derivation tree (also called Viterbi tree) in the hypergraph, or the k most probable trees. On a probabilistic hypergraph, we may also want to compute some expectations (e.g., expected translation length or feature expectation).

To perform the operations mentioned above, we usually use some dynamic programming (DP) algorithms. We can develop a dedicated program for each operation. For example, we may use a variant of the well-known Viterbi algorithm (Viterbi, 1967) to find the Viterbi tree. However, a more general framework to specify DP algorithms is semiring-weighted logic programming (Pereira and Warren, 1983; Shieber et al., 1994; Goodman, 1999; Eisner, Goldlust, and Smith, 2005; Lopez, 2009).[7] Under this framework, to perform

[7] In a nutshell, a semiring is a *set* with two operations and two identities. We write $\mathbf{K} = \langle K, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$

a different inference task, we just need to use a different semiring while other components in the framework remain unchanged. Goodman (1999) describes many useful semirings (e.g., Viterbi and k-best). While these semirings are mainly used at "testing" time, Eisner (2002) proposes an expectation semiring (for an FSA), which can be used for computing expectations that are useful for training (i.e., parameter estimation).

Our Contribution

We first extend the expectation semiring (Eisner, 2002), which was originally proposed for a finite-state machine, to a hypergraph. We then propose a novel *second-order* expectation semiring, nicknamed the "variance semiring."

The first-order expectation semiring allows us to efficiently compute a vector of first-order statistics (expectations; first derivatives) on the set of paths in an FSA or the set of trees in a hypergraph. The second-order expectation semiring *additionally* computes a matrix of second-order statistics (expectations of *products*; second derivatives (Hessian); derivatives of expectations).

We present details on how to compute many interesting quantities over the hypergraph using the expectation and variance semirings. These quantities include expected hypothesis length, feature expectation, entropy, cross-entropy, Kullback-Leibler divergence, Bayes risk, variance of hypothesis length, gradient of entropy and Bayes risk, covariance and Hessian matrix, and so on. The variance semiring is essential for many interesting training paradigms such as deterministic annealing (Rose, 1998), minimum risk (Smith and Eisner, 2006), active and semi-supervised learning (Grandvalet and Bengio, 2004; Jiao, Wang, Lee, Greiner, and Schuurmans, 2006). In these settings, we must compute the gradient of entropy or risk. The semirings can also be used for second-order gradient optimization algorithms.

1.4.2 Unsupervised Discriminative Training

Discriminative Training: Previous Work

Given a set of training examples (e.g., bilingual data in an MT task), the goal of a discriminative training method is to find an optimal weight vector θ^* where optimality is measured in terms of translation quality. This has been attempted in several works in the context of MT, by using various discriminative procedures such as *minimum error rate training* (Och, 2003; Macherey, Och, Thayer, and Uszkoreit, 2008), *averaged Perceptron* (Liang, Bouchard-Côté, Klein, and Taskar, 2006), *maximum conditional likelihood* (Blunsom, Cohn, and Osborne, 2008), *minimum risk* (Smith and Eisner, 2006; Li and Eisner, 2009), and *MIRA* (Watanabe, Suzuki, Tsukada, and Isozaki, 2007; Chiang, Knight, and

for a semiring with elements K, additive operation $\oplus$, multiplicative operation $\otimes$, additive identity $\mathbf{0}$, and multiplicative identity $\mathbf{1}$. As an example, the set of non-negative real numbers is a semiring $\langle \mathbb{R}_0^\infty, +, \times, 0, 1 \rangle$.

Wang, 2009). These procedures can significantly improve MT quality, because they directly optimize a performance metric and because they allow more complex models. However, one common drawback of such work is that it relies on the existence of high-quality in-domain supervised data (i.e., bilingual data), which is expensive to obtain, especially for a novel domain (e.g., web blog) or a low-resource language pair (e.g., between Urdu and English).

We present two unsupervised discriminative training methods: *minimum imputed-risk* and *contrastive language model estimation*, which rely on monolingual English data, but not its corresponding Chinese inputs.

Our Contribution: Minimum Imputed-Risk

When we only have monolingual English data, optimizing the performance of the translation system is a curious idea, since there is no Chinese input x to translate. Our solution is conceptually straightforward and relies on a reverse translation model (i.e., a model for English-to-Chinese translation). Specifically, we first guess the input x probabilistically from the observed output y using the reverse (English-to-Chinese) model. Then we train the discriminative Chinese-to-English system to do a "good job" at translating this imputed x back to y, in the sense of optimizing a given performance metric (e.g., BLEU (Papineni, Roukos, Ward, and Zhu, 2001)). Our method is theoretically sound and can be explained as minimizing imputed risk. Our method is also intuitive: it tries to ensure that probabilistic "round-trip" translation from the target-language sentence to the source language and back again will have low expected loss. Our experiments show that unsupervised discriminative training performs similarly to the supervised case, and often better. Also, augmenting supervised training with unsupervised data improves the performance.

Our Contribution: Contrastive Language Model Estimation

The minimum imputed-risk relies on a reverse translation model. We propose another unsupervised method, contrastive language model estimation, which can also exploit monolingual English data to perform discriminative training, but does not require a reverse system. It works as follows. We first extract a *confusion grammar* from a *bilingual grammar*. Specifically, whenever in the bilingual grammar we see two rules that have the same Chinese side (say "cn1") but two different English sides (say "eng1" and "eng2") , we will extract a confusion rule $X \rightarrow \langle \text{eng1}, \text{eng2} \rangle$. The confusion rule is English to English and captures the confusion that an MT system will have when translating the Chinese-side "cn1". Now, given a good English sentence $\tilde{y}$, we use the confusion grammar to produce many alternate English sentences y. This can be done as regular MT decoding as we can think that we are decoding the $\tilde{y}$ using an English-to-English "translation" system. The set of y generated can be thought as alternative translations generated by the SMT system if it had known the corresponding Chinese input x. Now, we can train a discriminative model on the generated data y (with the original English sentence $\tilde{y}$ as the training reference)

such that the original sentence $\tilde{y}$ will be highly liked by the model. The trained contrastive model can then be used as a regular language model for real MT decoding (e.g., translating Chinese to English).

Our experimental results show that the contrastive language model (CLM) performs better than a regular n-gram LM in terms of recovering an English sentence from its neighborhood (i.e., a set of alternative sentences that are generated from the English sentence). The CLM also improves the performance of an MT system.

1.4.3 Variational Decoding

Decoding: Previous Work

Given an input sentence x, an SMT system usually generates many possible translations y (along with the derivations d) that are encoded in a hypergraph. However, an end user may be interested in only a single translation for the input. So, the question is which one the system should present to the user. This is the decoding problem. In general, there are two kinds of decoding rules: maximum a posterior (MAP) and minimum Bayes risk (MBR) decoding.

Under the MAP decoding rule, the goodness of a translation string y is its posterior probability $p(y \mid x)$, which is the sum of the probabilities of all the derivations yielding x and y (see the formula of (1.5)). Since a hypergraph contains exponentially many translation strings and we need to carry out the above summation for each string, the MAP decoding problem turns out to be NP-hard as shown by Sima'an (1996) for a similar problem. Therefore, most systems (Koehn et al., 2003; Chiang, 2007) merely identify the single most probable derivation and report the corresponding string (i.e., the English yield of the derivation). This corresponds to a Viterbi approximation that measures the goodness of an output string using only its most probable derivation, ignoring all the others.

On the other hand, under the MBR decoding, the *badness* of a translation string y is its *expected* loss (or risk, see Chapter 2 for a definition), and we want to find a string that has minimum risk. In contrast to the MAP decoding, which chooses the translation with a maximum posterior probability, the MBR decoding will tend to choose a translation that most resembles the rest of the translations in the hypergraph. MBR decoding has been applied to MT for a k-best (Kumar and Byrne, 2004) or an FSA (Tromble, Kumar, Och, and Macherey, 2008) or a hypergraph (Kumar, Macherey, Dyer, and Och, 2009).

Our Contribution

As mentioned, MAP decoding under spurious ambiguity is intractable, and one usually uses a Viterbi approximation that ignores most of the derivations produced by the system. We propose a variational method that considers all the derivations but still allows tractable decoding. Given an input string, the original system produces a probability distribution p over possible output strings and their derivations. Our method constructs a second distribu-

tion $q \in \mathcal{Q}$ that approximates p as well as possible, and then finds the best string according to q. The last step is tractable because each $q \in \mathcal{Q}$ is defined (unlike p) without reference to the hidden derivations. Notice that q here does not approximate the entire translation process, but only the distribution over output strings *for a particular input*. This is why it can be a fairly good approximation even without looking at the hidden derivations.

In practice, we approximate with several different variational families $\mathcal{Q}$, corresponding to n-gram (Markov) models of different orders. We geometrically interpolate the resulting approximations q with one another (and with the original distribution p), justifying this interpolation as similar to the minimum-risk decoding for BLEU[8] proposed by Tromble et al. (2008). Experiments show that our approach improves the state of the art.

Our method should be applicable to collapsing spurious ambiguity for other tasks as well. Such tasks include data-oriented parsing (DOP), applications of Hidden Markov Models (HMMs) and mixture models, and other models with latent variables.

1.5 Roadmap

The dissertation is organized as follows. In Chapter 2, we will give a thorough review of the inference, decoding, and training methods that work for a hypergraph. The goal of this chapter is to give the readers who know little about MT (but have a good understanding on algorithms and statistical modeling in general) a global picture about the state of the art methods developed for hypergraphs in the context of MT. Readers who are familiar with these techniques can safely skip this chapter (although in later chapters we may refer back to the material presented in this chapter). The four chapters following Chapter 2 will present the main contributions of the dissertation. Specifically, Chapter 3 will present the first- and second-order expectation semirings that can be used to compute a large number of expectations over a hypergraph. These expectations will be useful both for the training and decoding methods presented in the three chapters followed right after Chapter 3. Readers who are less interested in computational efficiency can safely skip this chapter if they feel comfortable to assume that the expectations can be computed efficiently. The two chapters followed after will present the two unsupervised discriminative training methods, specifically, Chapter 4 for minimum imputed-risk, and Chapter 5 for contrastive language model estimation. Then, Chapter 6 will present the variation decoding, which should be quite independent from the other chapters. Finally, we will conclude in Chapter 7.

[8] BLEU (Papineni et al., 2001) is an automatic metric that measures how good a translation is compared with some reference translations.

Chapter 2

Inference, Decoding, and Training Methods over Hypergraphs: a Review

In this chapter, we present a review of the inference, decoding, and training methods that work on hypergraphs.[1] As mentioned, any finite-state automaton (FSA) can also be encoded as a hypergraph (in which every hyperedge is an ordinary edge that connects a *single* antecedent to a consequent).[2] Thus, the methods reviewed here apply directly to FSAs.

2.1 Definitions, Notations and Terminologies

Table 2.1 lists notation that will be used throughout the dissertation. Specifically, we use x to denote the input string in the source language, $\tilde{y}$ to denote the reference translation in the target language, y to denote any candidate translation considered by an SMT system, and y^* to denote the translation that is actually produced. We use $\mathrm{L}(y, \tilde{y})$ represents the loss of y if the true answer is $\tilde{y}$, and the loss used in this dissertation is the negated BLEU score (Papineni et al., 2001).

We use d to represent a derivation, and $\mathrm{X}(d)$ and $\mathrm{Y}(d)$ to denote its source- and target-language *yield*, respectively. We use $\mathrm{D}(x)$ (abbreviated as D when appropriate) to denote a set of derivations that are considered by the system for the input x, and $\mathrm{D}(x, y)$ to denote a subset of derivations that yield the input x and output y. We use $\mathrm{T}(x)$ to denote the set of all possible translation strings considered by the system.

For each input sentence x, we assume a hypergraph $\mathrm{HG}(x)$, which encodes a probability distribution over derivations (i.e., $p(d \mid x)$) and a distribution over translation strings (i.e., $p(y \mid x)$). Recall that a hypergraph is a pair $\langle V, E \rangle$, where V is a set of *nodes* (vertices) and E is a set of *hyperedges*, with each hyperedge connecting a *set* of *antecedent*

[1] Readers who are familiar with these techniques can safely skip this chapter (although the notations in Table 2.1 will be used in later chapters).

[2] Note that we assume the hypergraph generated is acyclic, which is true for most MT systems.

Notation	Meaning
x	input sentence
$\tilde{y}$	reference translation
y	any candidate translation considered by an SMT system
y^*	the translation produced by an SMT system
$\mathrm{L}(y, \tilde{y})$	the loss (e.g negated BLEU) incurred by producing y if the true answer is $\tilde{y}$
d	a derivation tree
$\mathrm{Y}(d)$	the target-language yield of d
$\mathrm{D}(x)$ or D	a set of derivations generated for x
$\mathrm{D}(x, y)$	a set of derivations that yield input x and output y
$\mathrm{T}(x)$	a set of translation strings generated for x
$\mathrm{HG}(x)$	a hypergraph generated for x
$T(e)$	a set of *antecedent nodes* of the hyperedge e
$I(v)$	a set of *incoming hyperedges* of node v

Table 2.1: Main notations used in the dissertation

nodes to a single *consequent node*. We write $T(e)$ to denote the set of antecedent nodes of a hyperedge e, and $I(v)$ for the set of *incoming hyperedges* of node v (i.e., hyperedges of which v is the consequent), which represent different ways of deriving v.

2.2 Atomic Inference Operations Over Hypergraphs

Given a weighted/probabilistic hypergraph, one might be interested in efficiently performing some atomic operations over it. In this section, we will give a list of such operations.

2.2.1 Counting Number of Derivations

We might be interested in knowing how many derivations are encoded in a hypergraph. For example, there are four derivations in the hypergraph of Figure 1.6(a). Knowing the number of derivations may give us a sense how big the search space is and thus how computationally expensive the search problem will be. Equivalently, given an FSA, we may be interested in counting the number of paths in it. In Section 2.3, we will show how to count the number of derivations/paths by using a counting semiring.

Graph	Topological	Best-first		
		no heuristic	with heuristic	with hierarchy
FSA	Viterbi	Dijkstra	A*	HA*
Hypergraph	CYK	Knuth	Klein and Manning	Generalized A^*

Table 2.2: Algorithms for extracting one-best from an FSA or hypergraph

2.2.2 Finding One-best Derivation (or Viterbi Derivation)

Very often, we are interested in finding the best derivation in the hypergraph where the goodness of a derivation is measured by some score, that is,

$$d^* = \operatorname*{argmax}_{d \in \mathrm{D}} \text{score}(d) \tag{2.1}$$

where score(d) is usually the model score assigned by θ as in (1.2) of page 9, although it can be any other metric of d (see the oracle extraction example later).

The problem above is the same as the lightest derivation problem defined by Knuth (1977) for a hypergraph. It is also equivalent to the shortest-path problem for an FSA (Dijkstra, 1959). Table 2.2 shows the classical algorithms that solve this problem. In general, the algorithms can be classified by whether the search follows a certain topological order or best-first. The well-known Viterbi (Viterbi, 1967) algorithm for an FSA and the Cocke-Younger-Kasami (CYK) algorithm (alternatively called CKY) for a hypergraph search the graph in a topological order.[3] In the best-first search category, we can further classify the algorithms by whether heuristic functions are used for estimating the cost from the current node to the goal node. The algorithms described by Dijkstra (1959) and Knuth (1977) are the classical ones without using a heuristic function, that is, they assume the lower bound of the cost from the current node to the goal node is always equal to zero. The well-known A* algorithm (Hart and Raphael, 1968) and its variant for a hypergraph (Klein and Manning, 2003) follow the latter category. The A* algorithm can be extended to a hierarchical version where multiple layers of heuristics are organized in a hierarchial manner. This is called hierarchial A* (HA*) (Holte, Perez, Zimmer, and Macdonald, 1996) or generalized A* for a hypergraph (Felzenszwalb and McAllester, 2007). All these algorithms are forward-chaining (meaning the search is from the ground truths to the goal node), but it is also possible to do backward-chaining (meaning the search starts from the goal node and then recursively derives its support) or a combination.

Oracle Translation/Tree

While a hypergraph represents a very large set of translations, it is quite possible that the reference translation is not contained in the hypergraph, due to pruning or the inherent

[3] Note that the CYK algorithm is also responsible for constructing the hypergraph itself.

deficiency in the translation model. In this case, we want to find the translation in the hypergraph that is most similar to the reference translation, with similarity computed by some metric such as BLEU. Such maximally similar translation will be called *oracle translation*, and the process of extracting it *oracle extraction*. In practice, we first extract an oracle *tree*, and then read its target-language yield as the oracle translation.[4] Formally, the oracle extraction problem can be defined as,

$$\begin{aligned} d^* &= \operatorname*{argmax}_{d \in \mathrm{D}} \quad \mathrm{score}(d) \\ &= \operatorname*{argmax}_{d \in \mathrm{D}} \quad -\mathrm{L}(\mathrm{Y}(d), \tilde{y}) \end{aligned}$$

where $\mathrm{L}(y', \tilde{y})$ is the loss (e.g., negated BLEU) of y' if the true answer is $\tilde{y}$. The oracle translation/tree will be useful for discriminative training as we will see later in Section 2.6. Dreyer, Hall, and Khudanpur (2007); Leusch, Matusov, and Ney (2008) present oracle extraction algorithms for an FSA while Li and Khudanpur (2009a) present a version for a hypergraph.

2.2.3 Finding k-best Derivations

In addition to one-best, we might also be interested in finding the k-best derivations. The k-best can be used to approximate the full hypothesis space when we perform sophisticated decoding (e.g., MBR decoding in Section 2.5) and discriminative training (e.g., MERT in Section 2.6), which it may be too expensive to perform over the full hypothesis space. It will also be useful for reranking where we can apply more sophisticated features on the k-best, but not on the full hypothesis space due to intractability in dynamic programming. Similarly, we might also be interested in finding k-best oracle translations. Mohri and Riley (2002) present efficient algorithms to find a k-best on an FSA, and Huang and Chiang (2005) present versions for a hypergraph. More recently, Pauls and Klein (2009) present an A* variant.

2.2.4 Finding Partition Functions, Expectations, and Gradients

In a weighted hypergraph, each derivation d has a weight. Essentially, it defines a function $p : \mathrm{D} \rightarrow \mathbb{R}_{\geq 0}$, where $p(d)$ specifies a probability distribution over the derivations in the hypergraph. This probability distribution may be unnormalized, from which we can obtain a normalized probability distribution by dividing $p(d)$ with a normalization constant (alternatively called the *partition function*), defined as,

$$Z \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) \tag{2.2}$$

[4] It is possible that multiple translations have the same BLEU scores. Even for a particular oracle translation, there might be many distinct oracle derivation trees yielding it, due to spurious ambiguity. In these cases, we break ties in an arbitrary manner.

In addition to $p(d)$, we may also be given functions of interest, say, $r, s : \mathrm{D} \to \mathbb{R}$, and we may be interested in computing the following expectations on the hypergraph HG:

$$\overline{r} \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) r(d) \tag{2.3}$$

$$\overline{s} \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) s(d) \tag{2.4}$$

$$\overline{t} \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) r(d) s(d) \tag{2.5}$$

For example, $r(d)$ (and $s(d)$) can be the length of the translation corresponding to derivation d. Then $\overline{r}/Z$ (and $\overline{s}/Z$) is the expected hypothesis length in the hypergraph, and the second-order statistic $\overline{t}/Z$ is the second moment of the length distribution, from which we can find the variance of hypothesis length since $\overline{t}/Z - (\overline{r}/Z)^2$. A more interesting expectation might be the entropy of the distribution $p(d)$, that is,

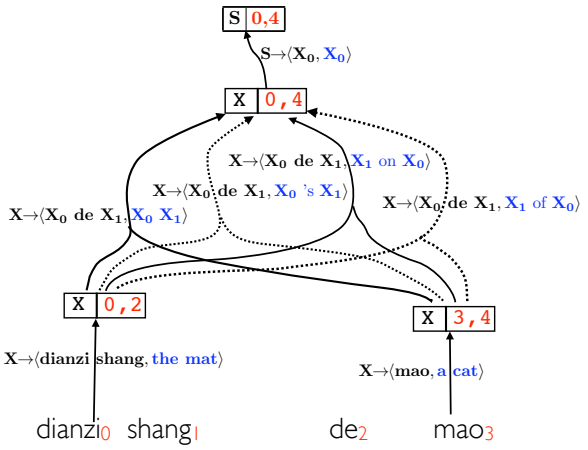
$$\mathrm{H}(p) = -\sum_{d \in \mathrm{D}} (p(d)/Z) \log(p(d)/Z)$$

Chapter 3 will show more examples of expectations, and show that the gradients with respect to the parameters θ that parameterize $p(d)$ are also expectations.

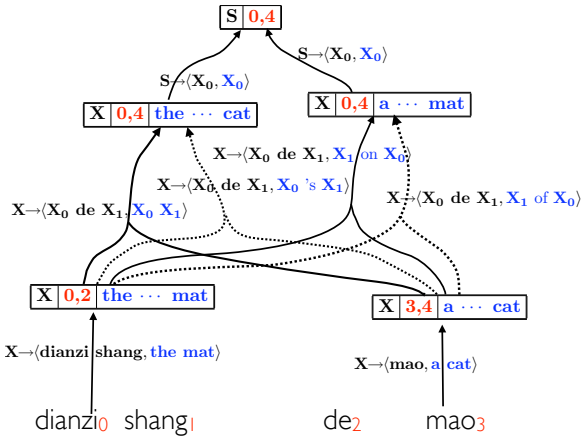
The normalization constant Z and the first-order expectations $\overline{r}$ and $\overline{s}$ can be found by using the classical forward-backward algorithm (Baum., 1972) for an FSA and the inside-outside algorithm (Baker, 1979) for a hypergraph. However, a more general way for computing them is to use an expectation semiring (Eisner, 2002) which was originally proposed for an FSA. We extend it to a hypergraph, and propose a novel second-order expectation semiring that can be used to additionally compute the second-order expectation $\overline{t}$. Please refer to Chapter 3 for details.

2.2.5 Integrating an n-gram model into a Hypergraph

All the operations above are for a single graph (e.g., a hypergraph or FSA). In some cases, we may need to perform an operation on more than one graph. For example, we may want to integrate an n-gram language model into a hypergraph (so that each derivation in the hypergraph will have a language model score, in addition to the scores assigned by other models). Figure 2.1 shows two hypergraphs, where the hypergraph of Figure 2.1(b) integrates a bigram language model (LM), but the one of Figure 2.1(a) does not. In the hypergraph without integrating an n-gram LM, the state information of a node includes the nonterminal symbol (e.g., X) and the source indices (e.g., (0,4)). In contrast, in a hypergraph with an n-gram LM, the state of a node should contain additional information to remember the $(n-1)$ words on the left (or right) corner of the node (these are called *n-gram states* or *language model states*). One way to view this is that a node in a hypergraph without integrating an n-gram model is *split* into many nodes after integrating an LM.



(a) A hypergraph without using a bigram LM. A node is identified by the nonterminal and source indices.



(b) A hypergraph integrated with a bigram LM. A node is identified by the nonterminal, source indices, and the word on the left/right corner of the node.

Figure 2.1: **Hypergraphs with/without integrating a bigram model.** The hypergraph of (b) integrates a bigram LM, while the one of (a) does not.

Clearly, the hypergraph becomes bigger after integrating an n-gram LM. In the example of Figure 2.1, the number of nodes increase from four to five, while the number of edges increases from seven to eight. The increase of the size will be much more substantial for a realistic hypergraph. In fact, this is a major challenge in making syntax-based MT decoding scalable. There are several ways to reduce the size of the hypergraph with an integrated LM. One is to collapse the n-gram states by exploiting the backoff structure of the LM as done by Li and Khudanpur (2008a). This does the *exact* integration. There are also *approximate* techniques such as cube-pruning (see Section 2.4.1).

More formally, we can think of integrating an n-gram model into a hypergraph as an *intersection* between a synchronous context free grammar (SCFG) and an finite state automaton (FSA). This can be illustrated as follows. A hypergraph (see Figure 2.1(a)) that is generated for a given input x and without integrating an n-gram model essentially represents a SCFG, except that the rules in the grammar are filtered by the input sentence x and are "instantiated" at a particular position. Moreover, an n-gram language model can be represented as an FSA (Allauzen, Mohri, and Roark, 2003). Therefore, integrating an n-gram model into a hypergraph correspond to an intersection operation between an SCFG and FSA.

2.3 Exact Inference over Hypergraphs: Semiring Parsing

One can implement a dedicated dynamic programming algorithm to solve each individual inference problem described above. In this section, we will present a more appealing framework: seimiring parsing (Goodman, 1999), which is a unified framework to describe such dynamic programming algorithms.

2.3.1 What is a semiring?

In a nutshell, a semiring is a *set* with two operations and two identities. We write $\mathbf{K} = \langle K, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ for a semiring with elements K, additive operation $\oplus$, multiplicative operation $\otimes$, additive identity $\mathbf{0}$, and multiplicative identity $\mathbf{1}$. As an example, the set of non-negative real numbers is a semiring $\langle \mathbb{R}_0^\infty, +, \times, 0, 1 \rangle$.

A semring has to satisfy some axiomatic properties. Namely, $\oplus$ is associative and commutative with identity $\mathbf{0}$, $\otimes$ is associative with two-sided identity $\mathbf{1}$, and $\otimes$ distributes over $\oplus$ from both sides.

2.3.2 Semiring-Weighted Inside Algorithm

To compute a particular quantity (e.g., number of derivations) on a hypergraph, one can follow the following recipe,

1. choose a semiring K
2. specify a "weight" $k_e \in K$ for each hyperedge e
3. run the inside algorithm (see Figure 2.2).

The quantity we are seeking will then be the inside weight at the root node (i.e, $\beta(\text{root})$), which is the *total weight* of all derivations in the hypergraph.

As shown in Figure 2.2, the $\otimes$ operation is used to obtain the weight of each derivation d by *multiplying* the weights of its component hyperedges e, that is, $k_d = \bigotimes_{e \in d} k_e$. The $\oplus$

Name	Semiring $\mathbf{K} = \langle K, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$	Purpose
Boolean	$\langle \{true, false\}, \vee, \wedge, false, true \rangle$	decide whether to accept a string
Viterbi score	$\langle \mathbb{R}_0^1, \max, \times, 0, 1 \rangle$	probability of best derivation
Inside score	$\langle \mathbb{R}_0^1, +, \times, 0, 1 \rangle$	probability of a string
Counting	$\langle \mathbb{R}_0^\infty, +, \times, 0, 1 \rangle$	count the number of derivations

Table 2.3: Several example semirings. These semirings can be used to compute different quantities on a hypergraph.

operation is used to *sum* over all derivations d in the hypergraph to obtain the *total* weight of the hypergraph HG, which is $\bigoplus_{d \in \mathrm{D}} k_d$.

As an example, to count the number of derivations in a hypergraph, one can use the so-called counting semiring (which is the set of *ordinary integers*), assign every hyperedge weight 1 (therefore each derivation also has weight 1), and run the inside algorithm of Figure 2.2. [5] The total weight returned by the algorithm will be the number of derivations. Figure 2.3 shows how the inside algorithm of Figure 2.2 works for this example.

In general, the total weight is a sum over exponentially many derivations d. But Figure 2.2 sums over these derivations in time only linear in the size of the hypergraph (i.e., number of hyperedges). Its correctness relies on axiomatic properties of the semiring. The distributive property is what makes Figure 2.2 work. The other properties are necessary to ensure that $\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d} k_e$ is well-defined.[6]

The algorithm in Figure 2.2 is general and can be applied with any semiring. Table 2.3 present several simple examples. Goodman (1999) shows several more semirings, e.g., Viterbi-derivation semiring (which computes not only the Viterbi score, but also the derivation itself) and k-best derivations semiring. In Chapter 3, we will introduce the first- and second-order expectation semirings, which can be used to compute a large number of expectations on a hypergraph.

2.4 Approximate Inference over Hypergraphs

In some cases, exact inference (e.g., the operations discussed in Section 2.2) may not be tractable or is just too slow. For example, as discussed in Section 2.2.5, intersecting an SCFG grammar (e.g., the Hiero grammar) with a finite state automaton (e.g., an n-gram

[5] Figure 2.2 shows how to compute the total weight of an acyclic hypergraph HG, where we assume the HG has *already* been built by deductive inference (Shieber et al., 1994). But in practice, the nodes' inside weights $\beta(v)$ are usually accumulated *as the hypergraph is being built*, so that pruning heuristics can consult them.

[6] Actually, the notation $\bigotimes_{e \in d} k_e$ assumes that $\otimes$ is commutative as well, as does the notation "for $u \in T(e)$" in our algorithms; neither specifies a loop order. One could however use a non-commutative semiring by ordering each hyperedge's antecedents and specifying that a derivation's weight is the product of the weights of its hyperedges *when visited in prefix order*.

INSIDE(HG, $\mathbf{K}$)

for v **in** topological order on HG ▷ each node
  ▷ find $\beta(v) \leftarrow \bigoplus_{e \in I(v)} (k_e \otimes (\bigotimes_{u \in T(e)} \beta(u)))$
  $\beta(v) \leftarrow \mathbf{0}$
  for $e \in I(v)$ ▷ each incoming hyperedge
    $k \leftarrow k_e$ ▷ hyperedge weight
    for $u \in T(e)$ ▷ each antecedent node
      $k \leftarrow k \otimes \beta(u)$
    $\beta(v) \leftarrow \beta(v) \oplus k$
return $\beta(\text{root})$

Figure 2.2: **Inside algorithm for an acyclic hypergraph HG**, which provides hyperedge weights $k_e \in \mathbf{K}$. This computes all "inside weights" $\beta(v) \in \mathbf{K}$, and returns $\beta(\text{root})$, which is total weight of the hypergraph, i.e., $\bigoplus_{d \in \text{D}} \bigotimes_{e \in d} k_e$.

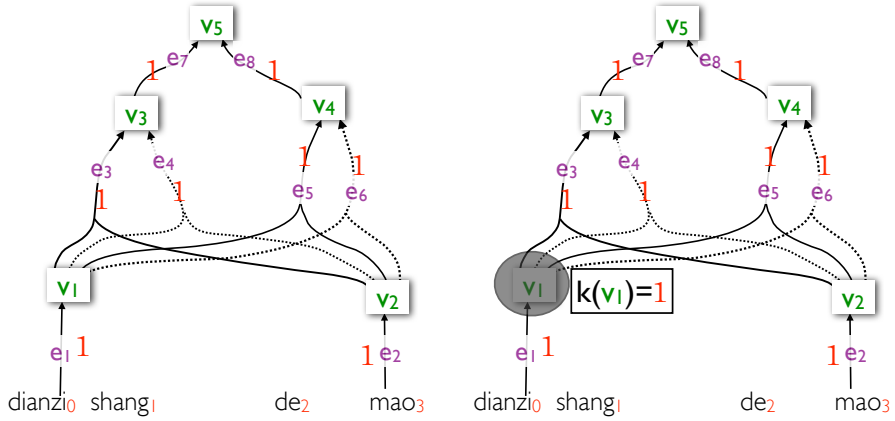
language model) is very slow. When exact inference is intractable, one has to resort to approximations. Below, we will review two flavors of approximations: heuristic-based and "principled"-based.

2.4.1 Heuristic-based Approximation

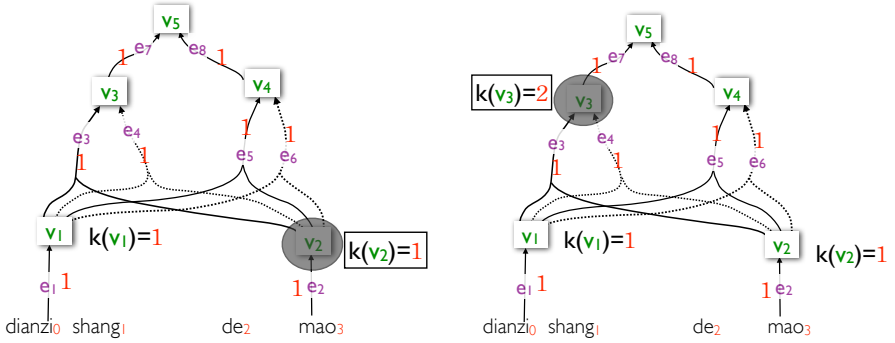
When exact inference is intractable for a particular task, a researcher who has a good understanding of the task will be able to identify key heuristics that are useful for finding approximate solutions. Below, we present two examples of heuristics that aim to speed up intersection between an SCFG and an n-gram model.

Cube-pruning

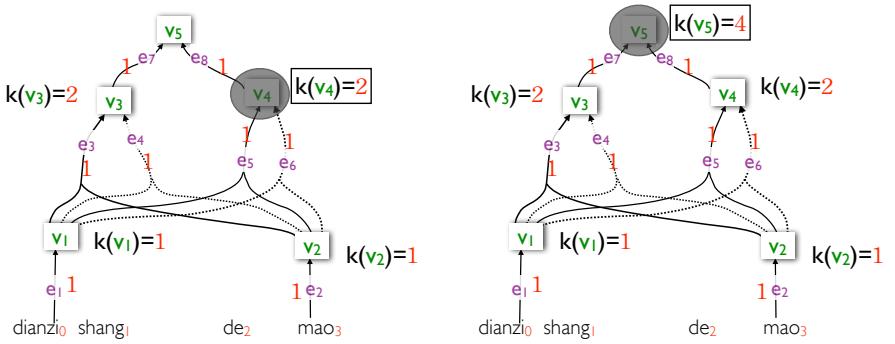
In parsing, a common operation is to combine small items (in the parse chart) to obtain bigger items by applying some rules. For example, the rule $X \rightarrow \langle X_0, X_1 \rangle$ can combine two small items X_0 and X_1 to get a bigger one. As mentioned in Section 2.2.5, when intersecting an SCFG with an n-gram model, the item will contain the n-gram states in addition to the nonterminal and source indices. Therefore, the number of such items will be very big. Also, there might be a large number of rules that can be used to combine items. Figure 2.4 shows that the space of combinations (of items using rules) form a cube. An exact intersection algorithm will compute all the possible combinations in the cube. This is too expensive. In contrast, the cube-pruning algorithm (Chiang, 2007) will explore a small corner of the cube. It works as follows. It first sorts the rules and the items along each axis of the cube (see Figure 2.4), then explores the combinations in a best-first manner, and stops whenever the cost of a new combination is worse than that of the best combination



(a) each edge is set with a weight of 1 (b) compute the weight at node v_1



(c) compute the weight at node v_2 (d) compute the weight at node v_3



(e) compute the weight at node v_4 (f) compute the weight at node v_5

Figure 2.3: These figures show how the inside algorithm from Figure 2.2 can be used to compute the number of trees by using a counting semiring.

explored so far by a margin. This is based on the heuristic that if an item falls outside the beam, then any item that would be generated due to a combination using a lower-scoring

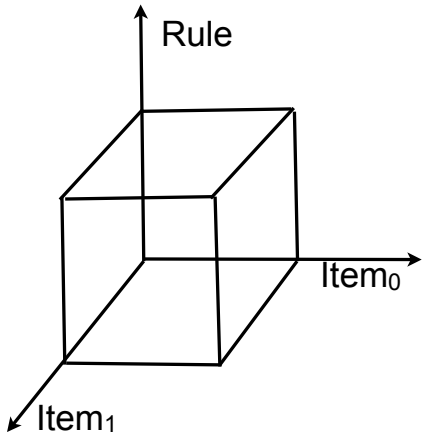


Figure 2.4: There are three axes (i.e., "Rule", "Item$_0$", and "Item$_1$") in parsing, and many possibilities along each axis. A rule will combine two small antecedent items to create a new bigger item. Such combinations form a cube.

rule or a lower-scoring antecedent item is also assumed to fall outside the beam. In this way, the vast majority of combinations gets pruned, and thus the intersection speeds up significantly. Huang and Chiang (2007) propose an extended version called cube-growing, and Hopkins and Langmead (2009) generalize cube-pruning as heuristic search.

Coarse To Fine Decoding

Another way to speed up the intersection between an SCFG and an n-gram model is the so-called coarse to fine decoding (Zhang and Gildea, 2008; Petrov, Haghighi, and Klein, 2008). It works as follows. Suppose we want to intersect an SCFG with a 5-gram LM. Instead of directly intersecting the SCFG with the 5-gram LM, we will first intersect the hypergraph with a unigram LM (for which we do not need to maintain any language model state). Then, we will run inside-outside on the hypergraph to compute the posterior/Viterbi probability of each hyperedge and node in the hypergraph, and prune out any hyperedge or node that has a low probability. Then, we will intersect the pruned hypergraph with a bigram LM. Again, we can prune the intersected hypergraph, after which we will integrate a trigram LM. This process continues until the 5-gram LM is integrated. The key intuition here is that if a hyperedge/node has a very low probability under a coarse model (e.g., a lower-order n-gram model) then it may not be worth considering it under the refined model (e.g., a high-order n-gram model).

2.4.2 "Principled" Approximations

In machine learning, there are several popular approaches to approximate inference when exact inference is intractable or too difficult. These methods are general and can be applied to a broad class of problems. They fall broadly into two classes: *stochastic* and *deterministic* (Bishop, 2006). Stochastic techniques such as Markov Chain Monte Carlo (MCMC) sampling are exact in the limit of infinite runtime, but tend to be too slow for large problems. By contrast, deterministic variational methods (Jordan, Ghahramani, Jaakkola, and Saul, 1999), including message-passing such as belief propagation (Minka, 2005), are inexact but scale up well. They approximate the original intractable distribution with one that factorizes better or has a specific parametric form (e.g., Gaussian).

Examples of using these techniques to solve difficult machine translation problems include MCMC sampling for maximum a posterior decoding and minimum Bayes risk decoding (Arun, Dyer, Haddow, Blunsom, Lopez, and Koehn, 2009), and variational inference for maximum a posterior decoding (Li, Eisner, and Khudanpur, 2009b). We will describe variational decoding in Chapter 6. Here we briefly review how to apply the MCMC sampling for MT. In a nutshell, it uses a sampler to draw some sample derivations from the hypergraph (that defines a probability distribution of derivations), and then perform inference on the samples (instead of on the original hypergraph). For example, we can find the Viterbi derivation in the samples, or compute some expectations on the samples. This is quite similar to the idea of using a k-best to approximate the full hypothesis space when performing inference (see Section 2.2.3). In particular, the samples can be thought as *randomized* k-"best" derivations.

Examples of using approximate inference to solve NLP problems other than MT includes belief propagation for dependency parsing (Smith and Eisner, 2008) and for morphology analysis (Dreyer and Eisner, 2009).

2.5 Decoding Methods over Hypergraphs

Given an input sentence x, an SMT system usually generates many possible translations y (along with their derivations d), which are encoded in a compact manner in a hypergraph. However, an end user may be interested in only a single translation of the input. So, the question is which translation the system should present to the user. This is the decoding problem. In general, there are two kinds of decoding rules: maximum a posteriori or its generalization (see footnote 8), minimum Bayes risk, as we will describe below.

2.5.1 Maximum A Posteriori (MAP) Decoding

The **maximum a posteriori (MAP)** decision rule is

$$y^* = \arg\max_{y \in \mathrm{T}(x)} p(y \mid x) \tag{2.6}$$

Under MAP, the goodness of a translation string y is its posterior probability $p(y \mid x)$, which we can obtain from $p(d \mid x)$ by marginalizing out d. Therefore, the MAP decision rule becomes

$$y^* = \arg\max_{y \in \mathrm{T}(x)} \sum_{d \in \mathrm{D}(x,y)} p(d \mid x) \tag{2.7}$$

where $\mathrm{D}(x, y)$ is the set of derivations d that yield x and y. Our derivation set $\mathrm{D}(x)$ is encoded in polynomial space, using a hypergraph or FSA. However, both $|\mathrm{D}(x)|$ and $|\mathrm{T}(x)|$ may be exponential in $|x|$. Since the marginalization needs to be carried out for each member of $\mathrm{T}(x)$, the decoding problem of (2.7) turns out to be NP-hard,[7] as shown by Sima'an (1996) for the similar problem of fending the most probable tree under the data oriented parsing (DOP) framework.

Viterbi Approximation

To approximate the intractable decoding problem of (2.7), most MT systems (Koehn et al., 2003; Chiang, 2007) use a simple **Viterbi** approximation,

$$y^* = \operatorname*{argmax}_{y \in \mathrm{T}(x)} \; p_{\text{Viterbi}}(y \mid x) \tag{2.8}$$

$$= \operatorname*{argmax}_{y \in \mathrm{T}(x)} \max_{d \in \mathrm{D}(x,y)} p(d \mid x) \tag{2.9}$$

$$= \mathrm{Y}\left(\operatorname*{argmax}_{d \in \mathrm{D}(x)} p(d \mid x) \right) \tag{2.10}$$

where $\mathrm{Y}(\cdot)$ denotes the yield of the Viterbi derivation. Clearly, (4.6) replaces the sum in (2.7) with a $\max$. In other words, it approximates the probability of a translation string by the probability of its most-likely derivation. (4.6) is found quickly via (2.10). The Viterbi approximation is simple and tractable, but it ignores most derivations.

k-best Approximation (or Crunching)

Another popular approximation enumerates the k best derivations in $\mathrm{D}(x)$, a set that we call $\mathrm{KD}(x)$. Modifying (2.7) to sum over only these derivations is called **crunching** by May and Knight (2006):

$$y^* = \operatorname*{argmax}_{y \in \mathrm{T}(x)} \; p_{\text{crunch}}(y \mid x) \tag{2.11}$$

$$= \operatorname*{argmax}_{y \in \mathrm{T}(x)} \sum_{d \in \mathrm{D}(x,y) \cap \mathrm{KD}(x)} p(d \mid x).$$

[7] Note that the marginalization for a particular y would be tractable; it is used at *training time* in certain training objective functions, e.g., maximizing the conditional likelihood of a reference translation (see Section 2.6).

Variational Decoding

The Viterbi and crunching methods approximate the intractable decoding of (2.7) by ignoring most of the derivations. In Chapter 6, we will present a novel variational approximation, which considers all the derivations but still allows tractable decoding. Essentially, it approximates the intractable distribution p in the MAP decoding with a simpler distribution q, and then uses q as a surrogate for decoding.

2.5.2 Minimum Bayes Risk (MBR) Decoding

In place of the MAP decoding, another commonly used decision rule is minimum Bayes risk (MBR):

$$y^* = \operatorname*{argmin}_y \mathrm{R}(y) \tag{2.12}$$

$$= \operatorname*{argmin}_y \sum_{y'} \mathrm{L}(y, y') p(y' \mid x) \tag{2.13}$$

where $\mathrm{L}(y, y')$ represents the loss of y if the true answer is y', and the **risk** of y is its expected loss.[8] Statistical decision theory shows MBR is optimal if $p(y' \mid x)$ is the true distribution, while in practice $p(y' \mid x)$ is given by a model at hand. In contrast to the MAP decoding, which chooses the translation with a maximum posterior probability, the MBR decoding will tend to choose a translation that most resembles the rest of the candidate translations.

Applying MBR to a hypergraph

While applying MBR to a k-best list, as done by Kumar and Byrne (2004), is relatively simple, its application to a compact data structure (e.g., FSA or hypergraph) is more involved. Tromble et al. (2008) describe a method on an FSA, while DeNero, Chiang, and Knight (2009); Kumar et al. (2009) apply it to a hypergraph.

The methods that work on a hypergraph have to use an approximate BLEU since the original BLEU is not linearly decomposable with respect to the hyperedges on the hypergraph. Tromble et al. (2008) propose the following loss function, of which a linear approximation to BLEU is a special case:

$$\mathrm{L}(y, \tilde{y}) = -(\theta_0 |y| + \sum_{w \in N} \theta_{|w|} c_w(y) \delta_w(\tilde{y})) \tag{2.14}$$

where w is an n-gram type, N is a set of n-gram types with $n \in [1, 4]$, $c_\theta(y)$ is the number of occurrences of the n-gram w in y, $\delta_\theta(\tilde{y})$ is an indicator function to check if $\tilde{y}$ contains at

[8] The MBR becomes the MAP decision rule of (2.6) if a so-called zero-one loss function is used: $\mathrm{L}(y, y') = 0$ if $y = y'$; otherwise $\mathrm{L}(y, y') = 1$.

least one occurrence of w, and θ_n is the weight that decides the relative importance of an n-gram match. If the hypergraph has already been annotated with n-gram ($n \geq 4$) language model states, the above loss function is **additively** decomposable and thus we are able to compute the risk (or expected loss).

2.6 Discriminative Training Methods over Hypergraphs

Given a set of training examples $(x_i, \tilde{y}_i)$ indexed by i. the training task is to find an optimal weight vector θ^* by maximizing/minimizing some objective function. Below, we first review the models we use, and then present different training methods along with their applications to a hypergraph.

2.6.1 Models

We have already defined the models in Section 1.3, and here we restate them for convenience of presentation.

Global Linear Model

We assume a model will assign a score to a derivation d as follows,

$$\text{score}(d) = f(d) \cdot \theta = \sum_j f_j(d)\theta_j \,, \tag{2.15}$$

where $f(d)$ is a feature vector depending on d (and its yields x and y). This is *a global linear model*, and can be used for non-probabilistic training methods such as MERT (Och, 2003), Structured Perceptron (Collins, 2002), and MIRA (Crammer, Dekel, Keshet, Shalev-Shwartz, and Singer, 2006).

Global Log-linear Model

We can also define a probabilistic model as follows,

$$p_\theta(d \mid x) = \frac{1}{Z_\theta(x)} e^{\gamma \cdot \text{score}(d)} \,, \tag{2.16}$$

where γ is a scaling factor to adjust the sharpness of the distribution (the larger γ is, the more peaky the distribution is), and $Z_\theta(x)$ is a normalization constant (or partition function) defined as,

$$Z_\theta(x) = \sum_{d \in \text{D}(x)} e^{\gamma \cdot \text{score}(d)} \,. \tag{2.17}$$

This is a *global log-linear model*, and can be applied to probabilistic training methods such as conditional random field (CRF) (Lafferty, McCallum, and Pereira, 2001) and minimum-risk training (Smith and Eisner, 2006).

Latent-variable Model

Recall that in machine translation, for a given observed pair $(x, \tilde{y})$, there might be many hidden/latent derivations d that yield x and $\tilde{y}$ due to spurious ambiguity. We can obtain the marginal distribution $p_\theta(\tilde{y} \mid x)$ as follows,

$$p_\theta(\tilde{y} \mid x) = \sum_{d \in \mathrm{D}(x, \tilde{y})} p_\theta(d \mid x)\,, \tag{2.18}$$

where $\mathrm{D}(x, \tilde{y})$ represents the set of derivation trees that yield x and $\tilde{y}$. This is a model with *a latent variable* (i.e., d), and can be applied to a probabilistic method.

2.6.2 Maximizing Conditional Likelihood

We can maximize the **conditional likelihood** (Lafferty et al., 2001) of the training examples as follows,

$$\theta^* \quad = \quad \arg\max_\theta \sum_i \log p_\theta(\tilde{y}_i \mid x_i) \tag{2.19}$$

This is commonly referred as a conditional random field (CRF). To address the issue of *overfitting*, we often add a *regularization term* (e.g, a Gaussian prior on θ) in the objective. Due to the use of the latent variable d, the objective function above is not convex, and thus the maximization may suffer from local-maximum problems.

To find the optimal θ^*, we can use a *gradient descent* method since the objective function is differentiable. The gradient of the log-likelihood (LL) of (2.19) with respect to the j-th parameter θ_j is,

$$\frac{\partial_{LL}}{\partial_{\theta_j}} = \sum_{i=1}^{N} \{ \sum_{d \in \mathrm{D}(x_i, \tilde{y}_i)} p_\theta(d \mid x_i) f_j(d) - \sum_{d \in \mathrm{D}(x_i)} p_\theta(d \mid x_i) f_j(d) \}\,. \tag{2.20}$$

Maximizing Conditional Likelihood on a Hypergraph

To apply the method above to a hypergraph, the main challenge is to compute the log-likelihood and its gradients on the hypergraph. These quantities can be computed by using an expectation semiring (as we shall see in Chapter 3).

Also, when the reference translation $\tilde{y}$ is not in the hypergraph (which is very likely in practice), we need to use an oracle translation as a surrogate. The oracle translation can be found by the algorithms mentioned in Section 2.2.2.

Perceptron$(x, \mathrm{D}(x), y)$

1 $\theta \leftarrow \vec{0}$ $\quad \triangleright$ initialize as zero vector
2 **for** $t \leftarrow 1$ **to** T
3 **for** $i \leftarrow 1$ **to** N
4 $d_i^* \leftarrow \arg\max_{d \in \mathrm{D}(x_i)} f(d) \cdot \theta$
5 **if** $(d_i^* \neq \tilde{d}_i)$
6 $\theta \leftarrow \theta + f(\tilde{d}_i) - f(d_i^*)$
7 **return** θ

Figure 2.5: The basic Perceptron algorithm

2.6.3 Average Perceptron

Unlike a CRF, the perceptron training (Collins, 2002) does not have an explicit objective to optimize.[9] Instead, it is a simple procedure that can learn optimal parameters. Specifically, given a set of training examples, the Perceptron algorithm *sequentially* iterates over the examples, and adjusts the parameter vector θ, as illustrated in Figure 2.5. In particular, whenever the one-best $d_i^* = \arg\max f(d) \cdot \theta$ under the current model θ is not the same as the reference derivation $\tilde{d}_i$, the parameter vector θ is incremented by the difference of the feature vectors between $\tilde{d}_i$ and d_i^*. After iterating over the training data a few times, an *averaged* model, $\overline{\theta} = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{N} \sum_{i=1}^{N} \theta_t^i$, is computed and is used for testing, where θ_t^i represents the parameter vector after seeing the i-th example in the t-th iteration, N represents the size of the training set, and T is the number of iterations of the Perceptron algorithm.

Applying Perceptron Training to a Hypergraph

To apply Perceptron training on a hypergraph, two algorithms are needed: oracle extraction and decoding. In particular, we need to find an oracle derivation tree $\tilde{d}_i$ when the reference translation is not in the hypergraph (see Section 2.2.2). Note that an oracle *tree*, instead of an oracle translation, is needed since Perceptron is not able to sum over latent derivations.

As indicated by line-4 of Figure 2.5, a decoding algorithm is needed. We can use the Viterbi decoding as in Figure 2.5. But, in general, any other decoding rule described in Section 2.5 can be employed as long as the decoder also produces a feature vector (in addition to the translation output). Perceptron training has been applied to MT for a k-best list (Liang et al., 2006) and a hypergraph (Li and Khudanpur, 2009b).

[9] However, Perceptron training could be viewed as stochastic gradient descent (SGD) optimization of the loss function of errors.

2.6.4 Minimum Error Rate Training

Intuitively, the CRF and the Perceptron are somewhat limited since they do not directly incorporate an MT metric (e.g., BLEU) in the training. Therefore, we may prefer to directly minimize the **one-best error** as in Och (2003),

$$\theta^* = \arg\min_{\theta} \sum_i \mathrm{L}(\mathrm{Y}(\arg\max_{d \in \mathrm{D}(x_i)} f(d) \cdot \theta),\ \tilde{y}_i) \tag{2.21}$$

$$\tag{2.22}$$

where $\mathrm{L}(y, \tilde{y})$ represents the **loss** of y if the true answer is $\tilde{y}$, and can be the negated BLEU score.[10] This objective matches the training with the test condition, i.e., finding a translation having minimum error with respect to the reference translation. However, it is piecewise constant and cannot be optimized using the general gradient descent method. Och (2003) developed a specialized line search to solve the optimization problem. This is not scalable when the model has a large number of parameters as the optimization needs to perform a line search along each dimension of the parameters, one by one.

Applying MERT to a Hypergraph

The original MERT (Och, 2003) procedure was proposed for a k-best list. Macherey et al. (2008) extend it to work for a FSA, while Kumar et al. (2009) further extend it to the case of a hypergraph. The main difficult computation for the latter two approaches is to find the upper envelope of the translation hypotheses (e.g., encoded in a hypergraph or FSA) for the line search (see Macherey et al. (2008) for details).

2.6.5 Minimizing Risk (MR)

Instead of minimizing the one-best error as in MERT (Och, 2003), we can also minimize the **risk** (or **expected loss**) on the training examples,

$$\theta^* = \arg\min_{\theta} \sum_i \mathrm{Risk}_\theta(\tilde{y}_i) \tag{2.23}$$

$$= \arg\min_{\theta} \sum_i \sum_y \mathrm{L}(y, \tilde{y}_i) p_\theta(y \mid x_i) \tag{2.24}$$

$$= \arg\min_{\theta} \sum_i \sum_{d \in \mathrm{D}(x_i)} \mathrm{L}(\mathrm{Y}(d), \tilde{y}_i) p_\theta(d \mid x_i). \tag{2.25}$$

[10] One might argue that an MT metric such as BLEU may not correlate perfectly well with the MT quality perceived by humans (Callison-Burch, Osborne, and Koehn, 2006b). However, that is an orthogonal issue. Any improvement of the MT metric itself can be directly reflected in the training objective. Future metrics should ideally be decomposed and quick to compute, in addition to strongly correlating with human judgements.

The objective above can be thought as a generalized version of two of the previous objectives. For example, as the scaling factor $\gamma \to +\infty$, the probability mass (see (2.16)) will be mostly on the one-best and thus the objective function becomes the minimization of the one-best error objective of (2.21). On the other hand, the minimum risk objective will resemble the maximum conditional likelihood objective (or CRF) of (2.19) if the following **zero-one** loss function is used in MR,

$$\mathrm{L}(y, \tilde{y}) = \begin{cases} 0 & \text{if } y = \tilde{y} \\ 1 & \text{otherwise.} \end{cases} \tag{2.26}$$

Applying MR to a Hypergraph

The objective function of (2.25) is differentiable and can be optimized using a general gradient descent method. It is relatively easy to compute the expected loss and its gradient if the hypothesis space is a k-best list as in Smith and Eisner (2006). In contrast, it is much more challenging to compute these quantities on a compact data structure such as a hypergraph or an FSA. In Chapter 3, we will show how to use a second-order expectation semiring to compute these quantities.

2.6.6 MIRA

MIRA (Crammer et al., 2006) is another method that can directly incorporate a loss metric into its objective. It is an online method. Given a current parameter vector θ' that is learnt from previous training examples, it updates the parameter vector to θ^* defined as follows,

$$\theta^* = \arg\min_{\theta} \frac{1}{2}||\theta - \theta'||^2 + C \sum_i \max_{d \in \mathrm{D}(x_i)} \{\mathrm{L}(\mathrm{Y}(d), \tilde{y}_i) - (f(\tilde{d}_i) - f(d)) \cdot \theta\} \tag{2.27}$$

where $\tilde{y}_i$ is an oracle translation and $\tilde{d}_i$ is the corresponding oracle derivation tree, and C is called capacity and can be tuned on a held out data set. The training example index i has the values corresponding to the indices of those examples in the current batch. The term under $\max$ corresponds to the *most violated constraint* (i.e., the maximum value by which the loss is greater than the margin $(f(\tilde{d}_i) - f(d)) \cdot \theta$). The above optimization problem can be solved using a variant of sequential minimal optimization (Platt, 1999).

Watanabe et al. (2007); Chiang, Marton, and Resnik (2008); Chiang et al. (2009) all apply the above method (or its variants) to MT. For efficiency, they use a set D(x) that is a k-best list (instead of a hypergraph). Moreover, Chiang et al. (2008, 2009) use not only a k-best list ranked by the current model θ', but also a k-best ranked by a combination of the model score and the loss (which they call loss-augmented inference).

Factor	Min-Risk	MERT	Perceptron	CRF	MIRA
Is scalable?	✓		✓	✓	✓
Integrates an MT metric?	✓	✓			✓
Handles latent variables?	✓			✓	
Does **not** need an oracle?	✓	✓			
Easy to regularize?	✓		✓	✓	✓

Table 2.4: Comparison of different training methods from different perspective

Applying MIRA to a Hypergraph

It is quite challenging to apply the above method to a hypergraph since we need to find the most violated constraint (i.e., the $\max$ in (2.27)) in the hypergraph for each possible value of θ. An alternative is to use the method described by Taskar, Chatalbashev, Koller, and Guestrin (2005), where we use dynamic programming to solve the optimization problem.

2.6.7 A Comparison of Training Methods

Table 2.4 gives a comparison for different methods from several perspectives.

Is the method scalable in tuning a large number of parameters?

All the methods except MERT are, in principle, scalable. To be more precise, online methods like MIRA and Perceptron will be the most scalable as the training generally involves finding the $\max$, CRF will be a little more expensive due to the computation of the partition function and its gradient (a first-order expectation), and minimum-risk will be even more expensive since we need to compute a second-order expectation.

Can the method integrate an MT metric (e.g., BLEU) during training?

While MERT, minimum-risk, and MIRA are able to incorporate an approximate BLEU (such as the one of (2.14)), CRF and Perceptron are not. The latter two approaches essentially use the zero-one loss function of (2.26), which may not be optimal for a task like MT as the reference translation is seldom in the (pruned) hypergraph.

Can the model handle latent variables (e.g., derivation trees)?

As shown by Blunsom et al. (2008), summing over the latent derivations as in (2.18) during training is very helpful for improving translation quality. CRF and minimum-risk use a probabilistic model, and thus can handle latent variables very easily (by marginalizing

out them during training). In contrast, non-probabilistic methods like MERT, Perceptron, and MIRA have no natural way to incorporate the latent variables.

Does the method require an oracle translation/tree when the reference is unreachable?

As mentioned before, in a realistic MT task, it is very likely that the reference translation is not in the hypergraph due to pruning or inherent deficiencies of the translation model. However, methods like Perceptron, CRF and MIRA require knowing the feature vector of the training target. Therefore, we have to use an oracle (along with its associated feature vector) as a surrogate. Finding such an oracle will be quite time consuming (see Li and Khudanpur (2009a)). Also the true quality (not in terms of BLEU score) of the oracle varies quite a lot. For example, a tiny change in the reference (which is used to guide the search of an oracle) may lead to a very different oracle. Therefore, we prefer a training method that does not rely on such arbitrary oracle. MERT and minimum-risk satisfy this requirement.

Can the model be regularized easily?

The goodness of a training method can be measured by how well the learnt model can perform on unseen test data. Therefore, regularizing the model during training is essential for any training method. It is very convenient to add an L2 regularization term on the weights in MIRA, CRF, and minimum-risk. Perceptron uses an averaged model for testing as mentioned before, which can be thought as model regularization. In contrast, it is not straightforward to perform regularization in MERT (although see Cer, Jurafsky, and Manning (2008) for an exploration).

2.6.8 Unsupervised Discriminative Training

While the above methods can significantly improve MT quality, because they directly optimize a performance metric (e.g., in MERT, minimum-risk, and MIRA) and because they allow more complex models (i.e., arbitrary features). However, one common drawback of such work is that it relies on the existence of high-quality in-domain supervised data (i.e., bilingual text), which is expensive to obtain, especially for a novel domain (e.g., weblogs) or a low-resource language pair (e.g., between Urdu and English). In Chapter 4, we will introduce a unsupervised method.

Chapter 3

First- and Second-order Expectation Semirings

Much of this chapter is based on Li and Eisner (2009). We first extend the expectation semiring (Eisner, 2002), which was originally proposed for an FSA, to a hypergraph. We then propose a novel *second-order* expectation semiring, nicknamed the "variance semiring." These semirings, within the framework of semiring parsing (see Section 2.3 on page 23), can be used to compute a large number of expectations on a hypergraph, including those that will be used in the following three chapters.

Specifically, the first-order expectation semiring allows us to efficiently compute a vector of first-order statistics (expectations; first derivatives) on the set of paths in an FSA or the set of trees in a hypergraph. The second-order expectation semiring *additionally* computes a matrix of second-order statistics (expectations of *products*; second derivatives (Hessian); derivatives of expectations).

We present details on how to compute many interesting quantities over the hypergraph using the expectation and variance semirings. These quantities include expected hypothesis length, feature expectation, entropy, cross-entropy, Kullback-Leibler divergence, Bayes risk, variance of hypothesis length, gradient of entropy and Bayes risk, covariance and Hessian matrix, and so on. The variance semiring is essential for many interesting training paradigms such as deterministic annealing (Rose, 1998), minimum risk (Smith and Eisner, 2006), active and semi-supervised learning (Grandvalet and Bengio, 2004; Jiao et al., 2006). In these settings, we must compute the gradient of entropy or risk. The semirings can also be used for second-order gradient optimization algorithms.

This chapter is organized as follows. In Section 3.1, we will present the expectation and variance semirings, and show how to use them to compute a single expectation. In Section 3.2, we will generalize them in computing a vector/matrix of expectations, and present speed-up tricks in these cases. We will then discuss how to compute gradients (which are essentially expectations) of several interesting functions in Section 3.3. In Section 3.4, we will present many example applications using the expectation and variance semirings. We will present some implementation details in Section 3.5, and summarize

this chapter in Section 3.6.

3.1 Finding Expectations on Hypergraphs

We now introduce the computational problems of this chapter and the semirings we use to solve them.

3.1.1 Problem Definitions

Recall that D represents the set of derivations encoded in a hypergraph. We are given a function $p : \mathrm{D} \to \mathbb{R}_{\geq 0}$, which decomposes **multiplicatively** over component hyperedges e of a derivation $d \in \mathrm{D}$: that is, $p(d) \stackrel{\text{def}}{=} \prod_{e \in d} p_e$. (For a particular way of parameterizing $p(d)$, please refer to Section 2.6.1 on page 31.) In practice, $p(d)$ will specify a probability distribution over the derivations in the hypergraph. It is often convenient to permit this probability distribution to be unnormalized, i.e., one may have to divide it through by some Z to get a proper distribution that sums to 1.

We are also given two functions of interest $r, s : \mathrm{D} \to \mathbb{R}$, each of which decomposes **additively** over its component hyperedges e: that is, $r(d) \stackrel{\text{def}}{=} \sum_{e \in d} r_e$, and $s(d) \stackrel{\text{def}}{=} \sum_{e \in d} s_e$.

We are now interested in computing the following quantities on the hypergraph HG:

$$Z \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) \tag{3.1}$$

$$\overline{r} \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) r(d) \tag{3.2}$$

$$\overline{s} \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) s(d) \tag{3.3}$$

$$\overline{t} \stackrel{\text{def}}{=} \sum_{d \in \mathrm{D}} p(d) r(d) s(d) \tag{3.4}$$

Note that $\overline{r}/Z$, $\overline{s}/Z$, and $\overline{t}/Z$ are expectations (under p) of $r(d)$, $s(d)$, and $r(d)s(d)$, respectively. More formally, the probabilistic interpretation is that D is a discrete sample space (consisting of all derivations in the hypergraph), p is a measure over this space, and $r, s : \mathrm{D} \to \mathbb{R}$ are random variables. Then $\overline{r}/Z$ and $\overline{s}/Z$ give the expectations of these random variables, and $\overline{t}/Z$ gives the expectation of their product $t = rs$, so that $\overline{t}/Z - (\overline{r}/Z)(\overline{s}/Z)$ gives their covariance.

Example 1: $r(d)$ is the length of the translation corresponding to derivation d (arranged by setting r_e to the number of target-side terminal words in the SCFG rule associated with e). Then $\overline{r}/Z$ is the expected hypothesis length.

Example 2: $r(d)$ evaluates the loss of d compared to a reference translation, using some additively decomposable loss function (e.g., the one of (2.14) on page 30). Then $\overline{r}/Z$ is the risk (expected loss), which is useful in minimum-risk training.

Example 3: $r(d)$ is the number of times a certain feature fires is presented in d. Then $\overline{r}/Z$ is the expected feature count, which is useful in maximum-likelihood training. We will generalize later in Section 3.2 to allow $r(d)$ to be a vector of features.

Example 4: Suppose $r(d)$ and $s(d)$ are identical and both compute hypothesis length. Then the second-order statistic $\overline{t}/Z$ is the second moment of the length distribution, so the variance of hypothesis length can be calculated as $\overline{t}/Z - (\overline{r}/Z)^2$.

3.1.2 Computing the Expectations

We will use the semiring parsing framework (see Section 2.3 on page 23) to compute the quantities (3.1)–(3.4). Although each is a sum over exponentially many derivations, we will compute it in $O(|\text{HG}|)$ time using the inside algorithm in Figure 3.1. Recall that the recipe (see Section 2.3.2 on page 23) to compute a quantity in the semiring framework is as follows: choose a semiring, specify a weight for each hyperedge, and run the inside algorithm.

In the simplest case, let the semiring $\mathbf{K} = \langle \mathbb{R}, +, \times, 0, 1 \rangle$, and define $k_e = p_e$ for each hyperedge e. Then the algorithm in Figure 3.1 reduces to the classical inside algorithm (Baker, 1979) and computes Z.

Next suppose $\mathbf{K}$ is the expectation semiring (Eisner, 2002), shown in Table 3.1. Define $k_e = \langle p_e, p_e r_e \rangle$. Then Figure 3.1 will return $\langle Z, \overline{r} \rangle$.

Finally, suppose $\mathbf{K}$ is our novel *second-order* expectation semiring, which we introduce in Table 3.2. Define $k_e = \langle p_e, p_e r_e, p_e s_e, p_e r_e s_e \rangle$. Then the algorithm of Figure 3.1 returns $\langle Z, \overline{r}, \overline{s}, \overline{t} \rangle$. Note that, to compute $\overline{t}$, one cannot simply construct a *first-order* expectation semiring by defining $t(d) \stackrel{\text{def}}{=} r(d)s(d)$ because $t(d)$, unlike $r(d)$ and $s(d)$, is not additively decomposable over the hyperedges in d.[1] Also, when $r(d)$ and $s(d)$ are identical, the second-order expectation semiring allows us to compute variance as $\overline{t}/Z - (\overline{r}/Z)^2$, which is why we may call our second-order expectation semiring the **variance semiring**.

3.1.3 Correctness of the Algorithms

To prove our claim about the first-order expectation semiring, we first observe that the definitions in Table 3.1 satisfy the semiring axioms. With a valid semiring, we then simply observe that the inside algorithm of Figure 3.1 returns the total weight $\bigoplus_{d \in \text{D}} \bigotimes_{e \in d} k_e$. So, we just need to show that the following equality is true,

$$\bigoplus_{d \in \text{D}} \bigotimes_{e \in d} k_e = \langle Z, \overline{r} \rangle. \tag{3.5}$$

To show this, we need first prove the following is true,

$$\bigotimes_{e \in d} k_e = \langle p(d), p(d)r(d) \rangle. \tag{3.6}$$

[1] However, in a more tricky way, the second-order expectation semiring can be constructed using the first-order expectation semiring, as will be seen in Section 3.2.3.

INSIDE(HG, **K**)

```
1  for v in topological order on HG    ▷ each node
2     ▷ find β(v) ← ⊕_{e∈I(v)} (k_e ⊗ (⊗_{u∈T(e)} β(u)))
3     β(v) ← 0
4     for e ∈ I(v)          ▷ each incoming hyperedge
5        k ← k_e            ▷ hyperedge weight
6        for u ∈ T(e)       ▷ each antecedent node
7           k ← k ⊗ β(u)
8        β(v) ← β(v) ⊕ k
9  return β(root)
```

Figure 3.1: Inside algorithm to compute all "inside weights" $\beta(v)$. Note that this is the same algorithm as in Figure 2.2 on page 24, and we repeat it here for convenience.

OUTSIDE(HG, **K**)

```
1  for v in HG
2     α(v) ← 0
3  α(root) ← 1
4  for v in reverse topological order on HG
5     for e ∈ I(v)          ▷ each incoming hyperedge
6        for u ∈ T(e)       ▷ each antecedent node
7           α(u) ← α(u) ⊕ (α(v) ⊗ k_e ⊗
8                          ⊗_{w∈T(e),w≠u} β(w))
```

Figure 3.2: Computes the "outside weights" $\alpha(v)$. Note that this can only be run after INSIDE(HG, **K**) of Figure 3.1 has already computed the inside weights $\beta(v)$.

This can be shown from the definitions of $\otimes, k_e, p(d)$, and $r(d)$. The main intuition is that $\otimes$ can be used to build up $\langle p(d), p(d)r(d)\rangle$ inductively from the k_e: if d decomposes into two disjoint subderivations d_1, d_2, then,

$$\begin{aligned}
&\langle p(d_1), p(d_1)r(d_1)\rangle \otimes \langle p(d_2), p(d_2)r(d_2)\rangle \\
&\qquad = \langle p(d_1)p(d_2), p(d_1)p(d_2)r(d_2) + p(d_2)p(d_1)r(d_1))\rangle && (3.7)\\
&\qquad = \langle p(d_1)p(d_2), p(d_1)p(d_2)(r(d_1) + r(d_2))\rangle && (3.8)\\
&\qquad = \langle p(d), p(d)r(d)\rangle. && (3.9)
\end{aligned}$$

Note that (3.7) follows from the definition of $\otimes$ of Table 3.1, while (3.9) follows from the definition of $p(d)$ and $r(d)$. The base cases are where d is a single hyperedge e, in which

INSIDE-OUTSIDE(HG, $\mathbf{K}$, X)

```
1  ▷ Run inside and outside on HG with only k_e weights
2  k̂ ← INSIDE(HG, K)        ▷ see Figure 3.1
3  OUTSIDE(HG, K)           ▷ see Figure 3.2
4  ▷ Do a single linear combination to get x̂
5  x̂ ← 0
6  for v in HG              ▷ each node
7      for e ∈ I(v)         ▷ each incoming hyperedge
8          k̄_e ← α(v)
9          for u ∈ T(e)     ▷ each antecedent node
10             k̄_e ← k̄_e β(u)
11         x̂ ← x̂ + (k̄_e x_e)
12 return ⟨k̂, x̂⟩
```

Figure 3.3: If every hyperedge specifies a weight $\langle k_e, x_e \rangle$ in some expectation semiring $\mathbb{E}_{\mathbf{K},X}$, then this inside-outside algorithm is a more efficient alternative to Figure 3.1 for computing the total weight $\langle \hat{k}, \hat{x} \rangle$ of the hypergraph, especially if the x_e are vectors. First, at lines 2–3, the inside and outside algorithms are run using only the k_e weights, obtaining only $\hat{k}$ (without $\hat{x}$) but also obtaining all inside and outside weights $\beta, \alpha \in \mathbf{K}$ as a side effect. Then the second component $\hat{x}$ of the total weight is accumulated in lines 5–11 as a linear combination of all the x_e values, namely $\hat{x} = \sum_e \overline{k_e} x_e$, where $\overline{k_e}$ is computed at lines 8–10 using α and β weights. The linear coefficient $\overline{k_e}$ is the "exclusive weight" for hyperedge e, meaning that the *product* $\overline{k_e} k_e$ is the total weight in $\mathbf{K}$ of all derivations $d \in \mathrm{D}$ that include e. Note that $\overline{k_e}$ itself does not include k_e as one might expect. Rather, x_e is defined to include k_e when computing expectations (see Section 3.1.2). However, when computing gradients, the definition of x_e does not have k_e (see Section 3.3).

case $\langle p(d), p(d)r(d) \rangle = k_e$ (thanks to our choice of k_e), and where d is empty, in which case $\langle p(d), p(d)r(d) \rangle = \mathbf{1}$. Therefore, it follows by induction that (3.6) holds.

Now, substitute (3.6) into (3.5), we obtain,

$$\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d} k_e = \bigoplus_{d \in \mathrm{D}} \langle p(d), p(d)r(d) \rangle \tag{3.10}$$

$$= \langle \sum_{d \in \mathrm{D}} p(d), \sum_{d \in \mathrm{D}} p(d)r(d) \rangle \tag{3.11}$$

$$= \langle Z, \overline{r} \rangle. \tag{3.12}$$

Note that (3.11) follows from the definition of $\oplus$ (in Table 3.1), while (3.12) follows from the definition of Z (of (3.1)), and $\overline{r}$ (of (3.2)). This completes the proof for the first-order expectation semiring.

Element	$\langle p, r \rangle$
$\langle p_1, r_1 \rangle \otimes \langle p_2, r_2 \rangle$	$\langle p_1 p_2,\ p_1 r_2 + p_2 r_1 \rangle$
$\langle p_1, r_1 \rangle \oplus \langle p_2, r_2 \rangle$	$\langle p_1 + p_2,\ r_1 + r_2 \rangle$
$\langle p, r \rangle^*$	$\langle p^*, p^* p^* r \rangle$
0	$\langle 0, 0 \rangle$
1	$\langle 1, 0 \rangle$

Table 3.1: **Expectation semiring**: Each element in the semiring is a **pair** $\langle p, r \rangle$. The second and third rows define the operations between two elements $\langle p_1, r_1 \rangle$ and $\langle p_2, r_2 \rangle$, and the last two rows define the identities. Note that the multiplicative identity **1** has an r component of 0.

Element	$\langle p, r, s, t \rangle$
$\langle p_1, r_1, s_1, t_1 \rangle \otimes \langle p_2, r_2, s_2, t_2 \rangle$	$\langle p_1 p_2,\ p_1 r_2 + p_2 r_1,\ p_1 s_2 + p_2 s_1,$ $p_1 t_2 + p_2 t_1 + r_1 s_2 + r_2 s_1 \rangle$
$\langle p_1, r_1, s_1, t_1 \rangle \oplus \langle p_2, r_2, s_2, t_2 \rangle$	$\langle p_1 + p_2,\ r_1 + r_2,\ s_1 + s_2,\ t_1 + t_2 \rangle$
$\langle p, r, s, t \rangle^*$	$\langle p^*,\ p^* p^* r,\ p^* p^* s,\ p^* p^* (p^* r s + p^* r s + t) \rangle$
0	$\langle 0, 0, 0, 0 \rangle$
1	$\langle 1, 0, 0, 0 \rangle$

Table 3.2: **Second-order expectation semiring** (variance semiring): Each element in the semiring is a **4-tuple** $\langle p, r, s, t \rangle$. The second and third rows define the operations between two elements $\langle p_1, r_1, s_1, t_1 \rangle$ and $\langle p_2, r_2, s_2, t_2 \rangle$, while the last two rows define the identities. Note that the multiplicative identity **1** has r,s and t components of 0.

The proof for the second-order expectation semiring is similar. In particular, one mainly needs to show that,

$$\bigotimes_{e \in d} k_e \quad = \quad \langle p(d), p(d)r(d), p(d)s(d), p(d)r(d)s(d) \rangle. \tag{3.13}$$

3.2 Generalizations and Speedups

In this section, we generalize beyond the case above where p, r, s are $\mathbb{R}$-valued. In general, p may be an element of some other semiring, and r and s may be **vectors** or other algebraic objects. When r and s are vectors, especially high-dimensional vectors, the basic "inside algorithm" of Figure 3.1 will be slow. We will show how to speed it up with an "inside-outside algorithm."

3.2.1 Allowing Feature Vectors and More

In general, for P, R, S, T, we can define the first-order expectation semiring $\mathbb{E}_{P,R} = \langle P \times R, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ and the second-order expectation semiring $\mathbb{E}_{P,R,S,T} = \langle P \times R \times S \times T, \oplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$, using the definitions from Tables 3.1–3.2. But do those definitions remain meaningful, and do they continue to satisfy the semiring axioms?

Indeed they do when $P = \mathbb{R}, R = \mathbb{R}^n, S = \mathbb{R}^m, T = \mathbb{R}^{n \times m}$, with rs defined as the outer product $rs^{\mathbf{T}}$ (a matrix) where $s^{\mathbf{T}}$ is the transpose of s. In this way, the second-order semiring $\mathbb{E}_{P,R,S,T}$ lets us take expectations of vectors and outer products of vectors. So we can find *means* and *covariances* of any number of linearly decomposable quantities (e.g., feature counts) defined on the hypergraph.

We will consider some other choices in Sections 3.2.3–3.2.4 below. Thus, for generality, we conclude this subsection by stating the precise technical conditions needed to construct $\mathbb{E}_{P,R}$ and $\mathbb{E}_{P,R,S,T}$:

- P is a semiring;
- R is a P-module (e.g, a vector space), meaning that it comes equipped with an associative and commutative addition operation with an identity element $\mathbf{0}$, and also a multiplication operation $P \times R \to R$, such that $p(r_1 + r_2) = pr_1 + pr_2$, $(p_1 + p_2)r = p_1 r + p_2 r$, and $p_1(p_2 r) = (p_1 p_2)r$;
- S and T are also P-modules;
- there is a multiplication operation $R \times S \to T$ that is bilinear, i.e., $(r_1 + r_2)s = r_1 s + r_2 s$, $r(s_1 + s_2) = rs_1 + rs_2$, $(pr)s = p(rs)$, and $r(ps) = p(rs)$.

As a matter of notation, note that above and in Tables 3.1–3.2, we overload "+" to denote any of the addition operations within P, R, S, T; overload "0" to denote their respective additive identities; and overload concatenation to denote any of the multiplication operations within or between P, R, S, T. "1" refers to the multiplicative identity of P. We continue to use distinguished symbols $\oplus, \otimes, \mathbf{0}, \mathbf{1}$ for the operations and identities in our "main semiring of interest," $\mathbb{E}_{P,R}$ or $\mathbb{E}_{P,R,S,T}$.

To compute equations (3.1)–(3.4) in this more general setting, we must still require multiplicative or additive decomposability, defining $p(d) \stackrel{\text{def}}{=} \prod_{e \in d} p_e$, $r(d) \stackrel{\text{def}}{=} \sum_{e \in d} r_e$, $s(d) \stackrel{\text{def}}{=} \sum_{e \in d} s_e$ as before. But the $\prod$ and $\sum$ operators here now denote appropriate operations within P, R, and S respectively (rather than the usual operations within $\mathbb{R}$).

3.2.2 Inside-Outside Speedup for First-Order Expectation Semirings

Under the first-order expectation semiring $\mathbb{E}_{\mathbb{R},\mathbb{R}^n}$, the inside algorithm of Figure 3.1 will return $\langle Z, \overline{r} \rangle$ where $\overline{r}$ is a vector of n feature expectations.

However, Eisner (2002, section 5) observes that this is inefficient when n is large. Why? The inside algorithm takes the trouble to compute an inside weight $\beta(v) \in \mathbb{R} \times \mathbb{R}^n$ for *each*

node v in the hypergraph (or FSA). The second component of $\beta(v)$ is a presumably *dense* vector of all features that fire in all subderivations rooted at node v. Moreover, as $\beta(v)$ is computed in lines 3–8 of Figure 3.1, that vector is built up (via the $\otimes$ and $\oplus$ operations of Table 3.1) as a linear combination of other dense vectors (the second components of the various $\beta(u)$). These vector operations can be slow.

A much more efficient approach (usually) is the traditional inside-outside algorithm (Baker, 1979).[2] Figure 3.3 generalizes the inside-outside algorithm to work with any expectation semiring $\mathbb{E}_{\mathbf{K},X}$.[3] We are given a hypergraph HG whose edges have weights $\langle k_e, x_e \rangle$ in this semiring. So now $k_e \in \mathbf{K}$ denotes only *part* of the edge weight, not all of it. More specifically, for the first-order expectation semiring, in Section 3.1.2 we have $k_e \stackrel{\text{def}}{=} \langle p_e, p_e r_e \rangle$, but now $k_e \stackrel{\text{def}}{=} p_e$ and $x_e \stackrel{\text{def}}{=} p_e r_e$. INSIDE-OUTSIDE$(\text{HG}, \mathbf{K}, X)$ finds $\bigoplus_{d \in \text{D}} \bigotimes_{e \in d} \langle k_e, x_e \rangle$, which has the form $\langle \hat{k}, \hat{x} \rangle$.

But, INSIDE$(\text{HG}, \mathbb{E}_{\mathbf{K},X})$ of Figure 3.1 could accomplish the same thing. So what makes the inside-outside algorithm (of Figure 3.3) more efficient? It turns out that $\hat{x}$ can be found quickly as a *single* linear combination $\sum_e \overline{k_e} x_e$ of just the feature vectors x_e that appear on *individual* hyperedges—typically a sum of very sparse vectors! And the linear coefficients $\overline{k_e}$, as well as $\hat{k}$, are computed entirely within the cheap semiring $\mathbf{K}$. They are based on β and α values obtained by first running INSIDE$(\text{HG}, \mathbf{K})$ (of Figure 3.1) and OUTSIDE$(\text{HG}, \mathbf{K})$ (of Figure 3.2), which use *only* the k_e part of the weights and ignore the more expensive x_e.

Why do we still need the expectation semiring?

It is noteworthy that the expectation semiring is not used at all by Figure 3.3. Although the return value $\langle \hat{k}, \hat{x} \rangle$ is in the expectation semiring, it is built up not by $\oplus$ and $\otimes$ but rather by computing $\hat{k}$ and $\hat{x}$ separately. One might therefore wonder why the expectation semiring and its operations (in Table 3.1) are still needed. One reason is that the input to Figure 3.3 consists of hyperedge weights $\langle k_e, x_e \rangle$ in the expectation semiring—and these weights may well have been constructed using $\otimes$ and $\oplus$. For example, Eisner (2002) uses finite-state operations such as composition, which do combine weights entirely within the expectation semiring before their result is passed to the forward-backward algorithm. A second reason is that when we work with a *second-order* expectation semiring in Section 3.2.4 below, the $\hat{k}, \beta$, and α values in Figure 3.3 will turn out to be elements of a first-order expectation semiring, and *they* must still be constructed by first-order $\otimes$ and $\oplus$, via calls to Figures 3.1–3.2.

[2] Note, however, that the expectation semiring requires *only* the forward/inside pass to compute expectations, and thus it is more efficient than the traditional inside-outside algorithm (which requires two passes) if we are interested in computing only a small number of quantities.

[3] This follows Eisner (2002), who similarly generalized the forward-backward algorithm.

Why does inside-outside work?

Whereas the inside algorithm computes $\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d} k_e$ in any semiring (e.g., the counting semiring), the inside-outside algorithm exploits the special structure of an expectation semiring. By that semiring's definitions of $\oplus$ and $\otimes$ (Table 3.1), the following is true,

$$\bigoplus_{d \in \mathrm{D}} \bigotimes_{e \in d} \langle k_e,\ x_e \rangle = \langle \sum_{d \in \mathrm{D}} \prod_{e \in d} k_e,\quad \sum_{d \in \mathrm{D}} \sum_{e \in d} (\prod_{e' \in d, e' \neq e} k_{e'}) x_e \rangle, \tag{3.14}$$

where the first component (giving $\hat{k}$) is found by calling the inside algorithm on just the k_e part of the weights. The second component (giving $\hat{x}$) can be rearranged into

$$\sum_{e} \sum_{d:\, e \in d} (\prod_{e' \in d, e' \neq e} k_{e'}) x_e = \sum_{e} \overline{k_e} x_e, \tag{3.15}$$

where $\overline{k_e} \stackrel{\text{def}}{=} \sum_{d:\, e \in d} (\prod_{e' \in d, e' \neq e} k_{e'})$ is found from β and α.

The application described at the start of this subsection is the classical inside-outside algorithm. Here $\langle k_e, x_e \rangle \stackrel{\text{def}}{=} \langle p_e, p_e r_e \rangle$, and the algorithm returns $\langle \hat{k}, \hat{x} \rangle = \langle Z, \overline{r} \rangle$. In fact, that $\hat{x} = \overline{r}$ can be seen directly as follows,

$$\overline{r} \;=\; \sum_{d} p(d) r(d) = \sum_{d} p(d) (\sum_{e \in d} r_e) = \sum_{e} \sum_{d:\, e \in d} p(d) r_e = \sum_{e} (\overline{k_e} k_e) r_e = \sum_{e} \overline{k_e} x_e = \hat{x}.$$

This uses the fact that $\overline{k_e} k_e = \sum_{d:\, e \in d} p(d)$.

3.2.3 Lifting Trick for Second-Order Semirings

We now observe that the second-order expectation semiring $\mathbb{E}_{P,R,S,T}$ can be obtained indirectly by nesting one first-order expectation semiring inside another! First "lift" P to obtain the first-order expectation semiring $\mathbf{K} \stackrel{\text{def}}{=} \mathbb{E}_{P,R}$. Then lift this a second time to obtain the "nested" first-order expectation semiring $\mathbb{E}_{\mathbf{K},X} = \mathbb{E}_{(\mathbb{E}_{P,R}),(S \times T)}$, where we equip $X \stackrel{\text{def}}{=} S \times T$ with the operations $\langle s_1, t_1 \rangle + \langle s_2, t_2 \rangle \stackrel{\text{def}}{=} \langle s_1 + s_2, t_1 + t_2 \rangle$ and $\langle p, r \rangle \langle s, t \rangle \stackrel{\text{def}}{=} \langle ps, pt + rs \rangle$. The resulting first-order expectation semiring has elements of the form $\langle \langle p, r \rangle, \langle s, t \rangle \rangle$. Table 3.3 shows that it is indeed isomorphic to $\mathbb{E}_{P,R,S,T}$, with corresponding elements $\langle p, r, s, t \rangle$.

This construction of the second-order semiring as a first-order semiring is a useful bit of abstract algebra, because it means that known properties of first-order semirings will also apply to second-order ones. First of all, we are immediately guaranteed that the second-order semiring satisfies the semiring axioms. Second, we can directly apply the inside-outside algorithm there, as we see below.

3.2.4 Inside-Outside Speedup for Second-Order Expectation Semirings

Given a hypergraph weighted by a *second-order* expectation semiring $\mathbb{E}_{P,R,S,T}$. By recasting this as the *first-order* expectation semiring $\mathbb{E}_{\mathbf{K},X}$ where $\mathbf{K} = \mathbb{E}_{P,R}$ and $X =$

$$
\begin{aligned}
\langle\langle p_1, r_1\rangle, \langle s_1, t_1\rangle\rangle \oplus \langle\langle p_2, r_2\rangle, \langle s_2, t_2\rangle\rangle &= \langle\langle p_1, r_1\rangle + \langle p_2, r_2\rangle,\ \langle s_1, t_1\rangle + \langle s_2, t_2\rangle\rangle \\
&= \langle\langle p_1 + p_2,\ r_1 + r_2\rangle,\ \langle s_1 + s_2,\ t_1 + t_2\rangle\rangle \\
\langle\langle p_1, r_1\rangle, \langle s_1, t_1\rangle\rangle \otimes \langle\langle p_2, r_2\rangle, \langle s_2, t_2\rangle\rangle &= \langle\langle p_1, r_1\rangle\langle p_2, r_2\rangle,\ \langle p_1, r_1\rangle\langle s_2, t_2\rangle + \langle p_2, r_2\rangle\langle s_1, t_1\rangle\rangle \\
&= \langle\langle p_1 p_2,\ p_1 r_2 + p_2 r_1\rangle,\ \langle p_1 s_2 + p_2 s_1,\ p_1 t_2 + p_2 t_1 + r_1 s_2 + r_2 s_1\rangle\rangle
\end{aligned}
$$

Table 3.3: **Constructing second-order expectation semiring as first-order**. Here we show that the operations in $\mathbb{E}_{\mathbf{K},X}$ are isomorphic to Table 3.2's operations in $\mathbb{E}_{P,R,S,T}$, provided that $\mathbf{K} \stackrel{\text{def}}{=} \mathbb{E}_{P,R}$ and $X \stackrel{\text{def}}{=} S \times T$ is a $\mathbf{K}$-module, in which addition is defined by $\langle s_1, t_1\rangle + \langle s_2, t_2\rangle \stackrel{\text{def}}{=} \langle s_1 + s_2, t_1 + t_2\rangle$, and left-multiplication by $\mathbf{K}$ is defined by $\langle p, r\rangle\langle s, t\rangle \stackrel{\text{def}}{=} \langle ps, pt + rs\rangle$.

$(S \times T)$, we can again apply INSIDE-OUTSIDE$(\text{HG}, \mathbf{K}, X)$ to find the total weight of all derivations. For example, to speed up the computation described in Section 3.1.2, we may define $\langle k_e, x_e\rangle = \langle\langle p_e, p_e r_e\rangle, \langle p_e s_e, p_e r_e s_e\rangle\rangle$ for each hyperedge e. Then the inside-outside algorithm of Figure 3.3 will compute $\langle \hat{k}, \hat{x}\rangle = \langle\langle Z, \overline{r}\rangle, \langle \overline{s}, \overline{t}\rangle\rangle$, more quickly than the inside algorithm of Figure 3.1 computed $\langle Z, \overline{r}, \overline{s}, \overline{t}\rangle$.

Figure 3.3 in this case will run the inside and outside algorithms *in the semiring* $\mathbb{E}_{P,R}$, so that $k_e, \hat{k}, \alpha, \beta$, and $\overline{k_e}$ will now be elements of $P \times R$ (not just elements of P as in the first-order case). Finally it finds $\hat{x} = \sum_e \overline{k_e} x_e$, where $x_e \in S \times T$.[4]

This is a particularly effective speedup over the inside algorithm when R consists of scalars (or small vectors) whereas S, T are sparse high-dimensional vectors. We will see exactly this case in our minimum-risk training in subsequent chapters of this dissertation, where our weights $\langle p, r, s, t\rangle$ denote (probability, risk, gradient of probability, gradient of risk), or (probability, entropy, gradient of probability, gradient of entropy).

[4] Figure 3.3 was already proved generally correct in Section 3.2.2. To understand more specifically how $\langle \overline{s}, \overline{t}\rangle$ gets computed, observe in analogy to the end of Section 3.2.2 that,

$$
\begin{aligned}
\langle \overline{s}, \overline{t}\rangle &= \textstyle\sum_d \langle p(d)s(d), p(d)r(d)s(d)\rangle \\
&= \textstyle\sum_d \langle p(d), p(d)r(d)\rangle\langle s(d), 0\rangle \\
&= \textstyle\sum_d \langle p(d), p(d)r(d)\rangle \sum_{e \in d} \langle s_e, 0\rangle \\
&= \textstyle\sum_e \sum_{d:\, e \in d} \langle p(d), p(d)r(d)\rangle\langle s_e, 0\rangle \\
&= \textstyle\sum_e (\overline{k_e} k_e)\langle s_e, 0\rangle \\
&= \textstyle\sum_e \overline{k_e}\langle p_e, p_e r_e\rangle\langle s_e, 0\rangle \\
&= \textstyle\sum_e \overline{k_e}\langle p_e s_e, p_e r_e s_e\rangle \\
&= \textstyle\sum_e \overline{k_e} x_e \\
&= \hat{x}.
\end{aligned}
$$

3.3 Finding Gradients on Hypergraphs

In sections 3.1.2 and 3.2.1, we saw how our semirings helped find the sum Z of all $p(d)$, and compute *expectations* $\overline{r}, \overline{s}, \overline{t}$ of $r(d)$, $s(d)$, and $r(d)s(d)$. It turns out that these semirings can *also* compute first- and second-order *partial derivatives* of all the above quantities, with respect to a parameter vector $\theta \in \mathbb{R}^m$. That is, we ask how they are affected when θ changes slightly from its current value. The elementary values p_e, r_e, s_e are now assumed to implicitly be functions of θ.

Case 1: Recall that $Z \stackrel{\text{def}}{=} \sum_d p(d)$ is computed by INSIDE(HG, $\mathbb{R}$) if each hyperedge e has weight p_e. "Lift" this weight to $\langle p_e, \nabla p_e \rangle$, where $\nabla p_e \in \mathbb{R}^m$ is a gradient vector. Now $\langle Z, \nabla Z \rangle$ will be returned by INSIDE(HG, $\mathbb{E}_{\mathbb{R},\mathbb{R}^m}$)— or, more efficiently, by INSIDE-OUTSIDE(HG, $\mathbb{R}, \mathbb{R}^m$).

Case 2: To differentiate a second time, we can "lift" the above weights again to obtain $\langle \langle p_e, \nabla p_e \rangle, \nabla \langle p_e, \nabla p_e \rangle \rangle = \langle \langle p_e, \nabla p_e \rangle, \langle \nabla p_e, \nabla^2 p_e \rangle \rangle$, where $\nabla^2 p_e \in \mathbb{R}^{m \times m}$ is the Hessian matrix of second-order mixed partial derivatives. These weights are in a second-order expectation semiring.[5] Now $\langle Z, \nabla Z, \nabla Z, \nabla^2 Z \rangle$ will be returned at the root node of a hypergraph by running INSIDE(HG, $\mathbb{E}_{\mathbb{R},\mathbb{R}^m,\mathbb{R}^m,\mathbb{R}^{m\times m}}$), or more efficiently by running INSIDE-OUTSIDE(HG, $\mathbb{E}_{\mathbb{R},\mathbb{R}^m}, \mathbb{R}^m \times \mathbb{R}^{m\times m}$).

Case 3: We may need to find expectations and their partial derivatives (e.g., in the minimum-risk training described in Section 2.6.5 on page 34). Recall that $\langle Z, \overline{r} \rangle$ is computed by INSIDE(HG, $\mathbb{E}_{\mathbb{R},\mathbb{R}^n}$) when the edge weights are $\langle p_e, p_e r_e \rangle$ with $r_e \in \mathbb{R}^n$. Lift these weights to $\langle \langle p_e, p_e r_e \rangle, \nabla \langle p_e, p_e r_e \rangle \rangle = \langle \langle p_e, p_e r_e \rangle, \langle \nabla p_e, (\nabla p_e) r_e + p_e (\nabla r_e) \rangle \rangle$. Now $\langle Z, \overline{r}, \nabla Z, \nabla \overline{r} \rangle$ will be returned at the root node by running INSIDE(HG, $\mathbb{E}_{\mathbb{R},\mathbb{R}^n,\mathbb{R}^m,\mathbb{R}^{n\times m}}$) or by INSIDE-OUTSIDE(HG, $\mathbb{E}_{\mathbb{R},\mathbb{R}^n}, \mathbb{R}^m \times \mathbb{R}^{n\times m}$).[6]

3.3.1 What Connects Gradients to Expectations?

In ***Case 1***, we claimed that the *same* algorithm will compute either gradients $\langle Z, \nabla Z \rangle$ or expectations $\langle Z, \overline{r} \rangle$, if the hyperedge weights are set to $\langle p_e, \nabla p_e \rangle$ or $\langle p_e, p_e r_e \rangle$ respectively. ***Cases 2–3*** relied on the fact that this relationship still holds even when the scalars $Z, p_e \in \mathbb{R}$ are replaced by more complex objects that we wish to differentiate. Our discussion below sticks to the scalar case for simplicity, but would generalize fairly straightforwardly. Pearlmutter and Siskind (2007) give the relevant generalizations of dual numbers.

[5] Modulo the trivial isomorphism from $\langle \langle p, r \rangle, \langle s, t \rangle \rangle$ to $\langle p, r, s, t \rangle$ (see Section 3.2.3), the intended semiring both here and in Case 3 is the one that was defined at the start of Section 3.2.1, in which r, s are vectors and their product is defined to be $rs^{\mathbf{T}}$, a matrix. However, when using this semiring to compute second derivatives (Case 2) or covariances, one may exploit the invariant that $r = s$, e.g., to avoid storing s and to compute $r_1 s_2 + s_1 r_2$ in multiplication simply as $2 \cdot r_1 r_2$.

[6] Or, if $n > m$, it is faster to instead use INSIDE-OUTSIDE(HG, $\mathbb{E}_{\mathbb{R},\mathbb{R}^m}, \mathbb{R}^n \times \mathbb{R}^{m\times n}$), swapping the second and third components of the 4-tuple and transposing the matrix in the fourth component. Algebraically, this changes nothing because $\mathbb{E}_{\mathbb{R},\mathbb{R}^n,\mathbb{R}^m\times\mathbb{R}^{n\times m}}$ and $\mathbb{E}_{\mathbb{R},\mathbb{R}^m,\mathbb{R}^n\times\mathbb{R}^{m\times n}}$ are isomorphic, thanks to symmetries in Table 3.2. This method computes the expectation of the gradient rather than the gradient of the expectation—they are equal.

This may seem wonderful and mysterious. We now show in two distinct ways why this follows from our setup of Section 3.1.1. At the end, we derive as a special case the well-known relationship between gradients and expectations in log-linear models.

From Expectations to Gradients

One perspective is that our semiring fundamentally finds expectations. Thus, we must be finding ∇Z by formulating it as a certain expectation $\bar{r}$. Specifically,

$$\nabla Z = \nabla \sum_d p(d) = \sum_d \nabla p(d) = \sum_d p(d) r(d) = \bar{r},$$

provided that $r(d) = (\nabla p(d))/p(d)$. That can be arranged by defining $r_e \stackrel{\text{def}}{=} (\nabla p_e)/p_e$, and that is why the input weights $\langle p_e, p_e r_e \rangle$ take the form $\langle p_e, \nabla p_e \rangle$. This can be proved as follows,

$$\begin{aligned} r(d) &= \sum_{e \in d} r_e = \sum_{e \in d} (\nabla p_e)/p_e && (3.16) \\ &= \sum_{e \in d} \nabla \log p_e = \nabla \sum_{e \in d} \log p_e = \nabla \log \prod_{e \in d} p_e && (3.17) \\ &= \nabla \log p(d) = (\nabla p(d))/p(d). && (3.18) \end{aligned}$$

From Gradients to Expectations

An alternative perspective is that our semiring fundamentally finds gradients. Indeed, pairs like $\langle p, \nabla p \rangle$ have long been used for this purpose (Clifford, 1873) under the name "dual numbers." Operations on dual numbers, including those in Table 3.1, compute a result in $\mathbb{R}$ *along with* its gradient. For example, our $\otimes$ multiplies dual numbers, since $\langle p_1, \nabla p_1 \rangle \otimes \langle p_2, \nabla p_2 \rangle = \langle p_1 p_2, p_1(\nabla p_2) + (\nabla p_1) p_2 \rangle = \langle p_1 p_2, \nabla(p_1 p_2) \rangle$.

The inside algorithm thus computes both Z and ∇Z in a single "forward" or "inside" pass—known as automatic differentiation in the forward mode. The inside-outside algorithm instead uses the reverse mode (a.k.a. back-propagation), where a separate "backward" or "outside" pass is used to compute ∇Z.

How can we modify this machinery to produce expectations $\bar{r}$ given some *arbitrary* r_e of interest? Automatic differentiation may be used on any function (e.g., a neural net), but for our simple sum-of-products function Z, it happens that $\nabla Z = \nabla(\sum_d \prod_e p_e) = \sum_d \sum_{e \in d} (\prod_{e' \in d, e' \neq e} p_{e'}) \nabla p_e$. Our trick is to surreptitiously *replace* the ∇p_e in the input weights $\langle p_e, \nabla p_e \rangle$ with $p_e r_e$. Then the output changes similarly: the algorithms will *instead* find $\sum_d \sum_{e \in d} (\prod_{e' \in d, e' \neq e} p_{e'}) p_e r_e$, which reduces to $\sum_d \sum_{e \in d} p(d) r_e = \sum_d p(d) \sum_{e \in d} r_e = \sum_d p(d) r(d) = \bar{r}$.

Log-linear Models as a Special Case

Replacing ∇p_e with $p_e r_e$ is unnecessary if ∇p_e already equals $p_e r_e$. That is the case in log-linear models, where $p_e \stackrel{\text{def}}{=} \exp(r_e \cdot \theta)$ for some feature vector r_e associated with e. So there, ∇Z already equals $\bar{r}$—yielding a key useful property of log-linear models, that $\nabla \log Z = (\nabla Z)/Z = \bar{r}/Z$, the vector of feature expectations (Lau, Rosenfeld, and Roukos, 1993).

3.4 Practical Applications

We are given a hypergraph HG whose hyperedges e are annotated with values p_e, and want to compute many quantities on it that are useful for different applications. Recall from Section 3.1.1 that the HG defines a probability distribution over all derivations d in the hypergraph, namely $p(d)/Z$ where $p(d) \stackrel{\text{def}}{=} \prod_{e \in d} p_e$.

3.4.1 First-Order Expectation Semiring

In Section 3.1, we show how to compute the **expected hypothesis length** or **expected feature counts**, using the algorithm of Figure 3.1 with a first-order expectation semiring $\mathbb{E}_{\mathbb{R},\mathbb{R}}$. In general, given hyperedge weights $\langle p_e, p_e r_e \rangle$, the algorithm computes $\langle Z, \bar{r} \rangle$ and thus $\bar{r}/Z$, the expectation of $r(d) \stackrel{\text{def}}{=} \sum_{e \in d} r_e$. We now show how to compute a few other quantities by choosing r_e appropriately.

Entropy on a Hypergraph

The entropy (more precisely, derivational entropy) of the distribution of *derivations* d in a hypergraph is[7]

$$\begin{aligned} \mathrm{H}_d(p) &= -\sum_{d \in \mathrm{D}} (p(d)/Z) \log(p(d)/Z) \\ &= \log Z - \frac{1}{Z} \sum_{d \in \mathrm{D}} p(d) \log p(d) \\ &= \log Z - \frac{1}{Z} \sum_{d \in \mathrm{D}} p(d) r(d) = \log Z - \frac{\bar{r}}{Z} \end{aligned} \tag{3.19}$$

provided that we define $r_e \stackrel{\text{def}}{=} \log p_e$ (so that $r(d) = \sum_{e \in d} r_e = \log p(d)$). Of course, we can compute $\langle Z, \bar{r} \rangle$ as explained in Section 3.1.2.

[7] Unfortunately, it is intractable to compute the entropy of the distribution over *strings* (each string's probability is a sum over several derivations). But in Section 6.3.4 on page 99 we will estimate the gap between derivational and string entropies.

Cross-Entropy and KL Divergence

We may be interested in computing the cross-entropy or KL divergence between two distributions p and q. For example, in variational decoding for machine translation (see Chapter 6), p is a distribution represented by a hypergraph, while q, represented by a finite state automaton, is an approximation to p. The cross entropy between p and q is defined as

$$\begin{aligned} \mathrm{H}(p, q) &= -\sum_{d \in \mathrm{D}} (p(d)/Z_p) \log (q(d)/Z_q) \\ &= \log Z_q - \frac{1}{Z_p} \sum_{d \in \mathrm{D}} p(d) \log q(d) \\ &= \log Z_q - \frac{1}{Z_p} \sum_{d \in \mathrm{D}} p(d) r(d) = \log Z_q - \frac{\overline{r}}{Z_p} \end{aligned} \tag{3.20}$$

where the first term Z_q can be computed using the inside algorithm with hyperedge weights q_e, and the numerator and denominator of the second term using an expectation semiring with hyperedge weights $\langle p_e, p_e r_e \rangle$ with $r_e \stackrel{\text{def}}{=} \log q_e$.

The KL divergence to p from q can be computed as $\mathrm{KL}(p \,\|\, q) = \mathrm{H}(p, q) - \mathrm{H}_d(p)$. As we will see in Chapter 6, the KL divergence can be used to measure how close the two distributions p and q are.

Expected Loss (Risk)

Given a reference sentence $\tilde{y}$, the expected loss (i.e., Bayes risk) of the hypotheses in the hypergraph is defined as,

$$\mathrm{Risk}(p) = \sum_{d \in \mathrm{D}} (p(d)/Z) \mathrm{L}(\mathrm{Y}(d), \tilde{y}) \tag{3.21}$$

where $\mathrm{Y}(d)$ is the target yield of d and $\mathrm{L}(y, \tilde{y})$ is the **loss** of the hypothesis y with respect to the reference $\tilde{y}$. The popular machine translation metric, BLEU (Papineni et al., 2001), is not **additively** decomposable, and thus we are not able to compute the expected loss for it. Instead, we can use the loss function defined in (2.14) on page 30, which is **additively** decomposable if the hypergraph is *already* annotated with n-gram ($n \geq 4$) language model states. Using $r_e \stackrel{\text{def}}{=} \mathrm{L}_e$ where L_e is the *loss* for a hyperedge e, we compute the expected loss,

$$\mathrm{Risk}(p) = \frac{\sum_{d \in \mathrm{D}} p(d) \mathrm{L}(\mathrm{Y}(d), \tilde{y})}{Z} = \frac{\overline{r}}{Z} \tag{3.22}$$

3.4.2 Second-Order Expectation Semirings

With second-order expectation semirings, we can compute from a hypergraph the expectation and variance of hypothesis length; the feature expectation vector and covariance

matrix; the Hessian (matrix of second derivatives) of Z; and the gradients of entropy and expected loss. The computations should be clear from earlier discussion. Below we compute gradient of entropy or Bayes risk.

Gradient of Entropy or Risk

It is easy to see that the gradient of entropy (3.19) is

$$\nabla \mathrm{H}_d(p) = \frac{\nabla Z}{Z} - \frac{Z\nabla\overline{r} - \overline{r}\nabla Z}{Z^2} \tag{3.23}$$

We may compute $\langle Z, \overline{r}, \nabla Z, \nabla\overline{r}\rangle$ as explained in ***Case 3*** of Section 3.3 by using $k_e \stackrel{\text{def}}{=} \langle p_e, p_e r_e, \nabla p_e, (\nabla p_e) r_e + p_e \nabla r_e\rangle \stackrel{\text{def}}{=} \langle p_e, p_e \log p_e, \nabla p_e, (1 + \log p_e)\nabla p_e\rangle$, where ∇p_e depends on the particular parameterization of the model. For example, in a log-linear model, $p_e \stackrel{\text{def}}{=} e^{f(e)\cdot\theta}$ and thus $\nabla p_e = p_e f(e)$, where θ is a parameter vector and $f(e)$ is a feature vector depending on the hyperedge e.

Similarly, the gradient of risk of (3.22) is

$$\nabla \mathrm{Risk}(p) = \frac{Z\nabla\overline{r} - \overline{r}\nabla Z}{Z^2} \tag{3.24}$$

We may compute $\langle Z, \overline{r}, \nabla Z, \nabla\overline{r}\rangle$ using $k_e \stackrel{\text{def}}{=} \langle p_e, p_e \mathrm{L}_e, \nabla p_e, \mathrm{L}_e \nabla p_e\rangle$.

3.4.3 Summary of the Applications

Table 3.4 summarizes the list of quantities we can compute using first- and second-order expectation semirings. For each quantity, it specifies the weight (i.e., k_e) for a hyperedge e, the weight (i.e., k_{root}) returned at the root node after running the inside algorithm of Figure 3.1 (or the inside-outside speedup of Figure 3.3), and the final value we should use for the quantity we are seeking.

3.5 Implementation Details

3.5.1 Preventing Underflow/Overflow

In Tables 3.1–3.2, we do not discuss how to store p, r, s, and t. If p is a probability, it often suffers from the underflow problem. r, s, and t may suffer from both underflow and overflow problems, depending on their scales.

To address these, we could represent p in the $\log$ domain as usual. However, r, s, and t can be positive or negative, and we cannot directly take the $\log$ of a negative number. Therefore, we represent real numbers as ordered pairs. Specifically, to represent $a = s_a e^{\ell_a}$, we store $\langle s_a, \ell_a\rangle$, where the $s_a \in \{+, -\}$ is the **sign bit** of a and the floating-point number ℓ_a is the **natural logarithm** of $|a|$.[8] Table 3.5 shows the "$\cdot$" and "$+$"operations.

[8] An alternative that avoids log and exp is to store $a = f_a 2^{e_a}$ as $\langle f_a, e_a\rangle$, where f_a is a *floating-point*

Quantity	k_e	k_{root}	**Final**
Expectation	$\langle p_e, p_e r_e \rangle$	$\langle Z, \overline{r} \rangle$	$\overline{r}/Z$
Entropy	$r_e \stackrel{\text{def}}{=} \log p_e$, so $k_e = \langle p_e, p_e \log p_e \rangle$	$\langle Z, \overline{r} \rangle$	$\log Z - \overline{r}/Z$
Cross-entropy	$\langle q_e \rangle$ $r_e \stackrel{\text{def}}{=} \log q_e$, so $k_e = \langle p_e, p_e \log q_e \rangle$	$\langle Z_q \rangle$ $\langle Z_p, \overline{r} \rangle$	$\log Z_q - \overline{r}/Z_p$
Bayes risk	$r_e \stackrel{\text{def}}{=} \mathrm{L}_e$, so $k_e = \langle p_e, p_e \mathrm{L}_e \rangle$	$\langle Z, \overline{r} \rangle$	$\overline{r}/Z$
First-order gradient	$\langle p_e, \nabla p_e \rangle$	$\langle Z, \nabla Z \rangle$	∇Z
Covariance matrix	$\langle p_e, p_e r_e, p_e s_e, p_e r_e s_e \rangle$	$\langle Z, \overline{r}, \overline{s}, \overline{t} \rangle$	$\frac{\overline{t}}{Z} - \frac{\overline{r}\,\overline{s}^{\mathbf{T}}}{Z^2}$
Hessian matrix	$\langle p_e, \nabla p_e, \nabla p_e, \nabla^2 p_e \rangle$	$\langle Z, \nabla Z, \nabla Z, \nabla^2 Z \rangle$	$\nabla^2 Z$
Gradient of expectation	$\langle p_e, p_e r_e, \nabla p_e, (\nabla p_e) r_e + p_e (\nabla r_e) \rangle$	$\langle Z, \overline{r}, \nabla Z, \nabla \overline{r} \rangle$	$\frac{Z \nabla \overline{r} - \overline{r} \nabla Z}{Z^2}$
Gradient of entropy	$\langle p_e, p_e \log p_e, \nabla p_e, (1 + \log p_e) \nabla p_e \rangle$	$\langle Z, \overline{r}, \nabla Z, \nabla \overline{r} \rangle$	$\frac{\nabla Z}{Z} - \frac{Z \nabla \overline{r} - \overline{r} \nabla Z}{Z^2}$
Gradient of risk	$\langle p_e, p_e \mathrm{L}_e, \nabla p_e, \mathrm{L}_e \nabla p_e \rangle$	$\langle Z, \overline{r}, \nabla Z, \nabla \overline{r} \rangle$	$\frac{Z \nabla \overline{r} - \overline{r} \nabla Z}{Z^2}$

Table 3.4: **A summary table of the quantities that can be computed using first- and second-order expectation semirings.** For each quantity, the table specifies the weight (i.e., k_e) for each hyperedge e, the weight (i.e., k_{root}) returned at the root node after running the inside algorithm of Figure 3.1 (or the inside-outside speedup of Figure 3.3), and the final value we should use for the quantity. In the column of "Quantity", we **boldface** those quantities that are general, while the quantities not boldfaced are specific examples of the general quantity above (e.g., *entropy* is an example of **expectation**). The form of p_e and ∇p_e depends on the particular parameterization of the model. For example, in a log-linear model, $p_e \stackrel{\text{def}}{=} e^{f(e) \cdot \theta}$ and thus $\nabla p_e = p_e f(e)$, where θ is a parameter vector and $f(e)$ is a feature vector depending on the hyperedge e.

3.5.2 Implementation Guide

While the presentation of this chapter might look quite dense and technical, the implementation may not be that difficult. In general, there are three kinds of concepts that need to be instantiated in the implementation: *algorithms*, *semirings*, and *applications*.

To use a semiring to compute a quantity, we need to at least implement the inside algo-

number and e_a is a sufficiently wide *integer*. E.g., combining a 32-bit f_a with a 32-bit e_a will in effect extend f_a's 8-bit internal exponent to 32 bits by adding e_a to it. This gives much more dynamic range than the 11-bit exponent of a 64-bit double-precision floating-point number, if vastly less than in Table 3.5.

s_a	s_b	$a+b$		$a \cdot b$	
		s_{a+b}	ℓ_{a+b}	$s_{a \cdot b}$	$\ell_{a \cdot b}$
+	+	+	$\ell_a + \log(1 + e^{\ell_b - \ell_a})$	+	$\ell_a + \ell_b$
+	-	+	$\ell_a + \log(1 - e^{\ell_b - \ell_a})$	-	$\ell_a + \ell_b$
-	+	-	$\ell_a + \log(1 - e^{\ell_b - \ell_a})$	-	$\ell_a + \ell_b$
-	-	-	$\ell_a + \log(1 + e^{\ell_b - \ell_a})$	+	$\ell_a + \ell_b$

Table 3.5: **Storing signed values in log domain**: each value a $(= s_a e^{\ell_a})$ is stored as a **pair** $\langle s_a, \ell_a \rangle$ where s_a and ℓ_a are the **sign bit** of a and **natural logarithm** of $|a|$, respectively. This table shows the operations between two values $a = s_a 2^{\ell_a}$ and $b = s_b 2^{\ell_b}$, assuming $\ell_a \geq \ell_b$. *Note:* $\log(1 + x)$ (where $|x| < 1$) should be computed by the Mercator series $x - x^2/2 + x^3/3 - \cdots$, e.g., using the math library function log1p.

rithm of Figure 3.1, which works for any semiring (including the expectation and variance semirings). If we need to compute expectations and use the speed-up trick described in Section 3.2, we also need to implement the outside algorithm of Figure 3.2 (which requires the inside algorithm), and the inside-outside algorithm of Figure 3.3 (which will call both the inside and outside algorithms as sub-routines).

In terms of semirings, if we do not use the inside-outside speedup, we need to implement the operations both for expectation semiring (see Table 3.1) and for variance semiring (see Table 3.2). In contrast, if we use the speedup, we just need to implement the operations for expectation semiring in Table 3.1, and also implement a multiplication operation for $\overline{k_e} x_e$ on line-11 of Figure 3.3. For the second-order expectation semiring, such a product operation is defined in Section 3.2.3. In any case, to store the semiring values in log-domain, we need to implement Table 3.5.

For specific applications, we can follow the semiring parsing recipe described in Section 2.3.2 on page 23: choose a semiring, specify a weight for each edge, and run the inside algorithm (or the inside-outside for speedup). Table 3.4 gives many examples on choosing a semiring and specifying a weight for computing many different expectations.

For an example of implementation, please see the open-source software **Joshua** (Li, Callison-Burch, Dyer, Ganitkevitch, Khudanpur, Schwartz, Thornton, Weese, and Zaidan, 2009a).

3.6 Summary

We presented first-order expectation semirings and inside-outside computation, and developed extensions to higher-order expectation semirings. This enables efficient computation of many interesting quantities over the exponentially many derivations encoded in a hypergraph: second derivatives (Hessian), expectations of products (covariances), and expectations such as risk and entropy along with their derivatives. To our knowledge, algo-

rithms for these problems have not been presented before.

Our approach is theoretically elegant, like other work in this vein (Goodman, 1999; Lopez, 2009; Gimpel and Smith, 2009). We used it practically to enable a new form of minimum-risk training in the next chapters. Our implementation was released within the open-source MT toolkit **Joshua** (Li et al., 2009a).

Chapter 4

Minimum Imputed Risk Training

In this chapter, we describe an **unsupervised** method for discriminative training. Our method assumes that we are given some output data $\tilde{y}$ (e.g., English sentences), but not the corresponding input data x (e.g., Chinese sentences). It also assumes that we have a reasonably good **reverse** conditional model $p_\phi(x \mid \tilde{y})$ parameterized by ϕ. Our goal is to learn a good **forward** conditional model $p_\theta(y \mid x)$ by tuning θ.

In the case of an MT task, our method works as follows. First guess x probabilistically from the observed $\tilde{y}$ using a reverse (English-to-Chinese) model. Then train the discriminative Chinese-to-English system to do a "good job" at translating this imputed x back to $\tilde{y}$, in the sense of optimizing the given performance metric.

Our method is theoretically sound and can be explained as minimizing imputed risk. Our method is also intuitive: it tries to ensure that probabilistic "round-trip" translation (from the target-language sentence to the source language and back again) will have low expected loss.

We perform experiments by using the open-source MT toolkit **Joshua** (Li et al., 2009a). Our experiments show that unsupervised discriminative training performs similarly to (and often better than) the supervised case. Also, adding unsupervised data into the supervised training improves the performance.

4.1 Minimum Empirical Risk (for Supervised Discriminative Training)

Let us first review discriminative training in the *supervised* setting—as used in MERT (Och, 2003) and other methods discussed in Section 2.6 on page 30.

One wishes to tune the parameters θ of some complex translation system $\delta_\theta(x)$. The function δ_θ, which translates Chinese x to English y, may have any form and need not be probabilistic. For example, the parameters θ may define a scoring function along with pruning and decoding heuristics for extracting a high-scoring translation y.

The goal of discriminative training is to choose θ to minimize the expected loss of this function, as defined by a given task-specific **loss function** $\mathrm{L}(y, \tilde{y})$ that measures how bad it would be to output y when the correct output is $\tilde{y}$. (For example, for an MT system that will be judged by the BLEU metric (Papineni et al., 2001), the loss might be a negated BLEU score.) To be precise, we hope to find θ with low **Bayes risk**, that is,[1]

$$\theta^* = \arg\min_\theta \sum_{x,y} p(x,y)\mathrm{L}(\delta_\theta(x), y) \tag{4.1}$$

where $p(x, y)$ is the true distribution over (input,output) pairs.[2]

Of course $p(x, y)$ is not known. In practice, one typically does **empirical risk** minimization. This means replacing $p(x, y)$ above with the empirical distribution $\tilde{p}(x, y)$ given by the supervised training set. Therefore,

$$\begin{aligned} \theta^* &= \arg\min_\theta \sum_{x,y} \tilde{p}(x,y)\mathrm{L}(\delta_\theta(x), y) \\ &= \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathrm{L}(\delta_\theta(x_i), \tilde{y}_i) \end{aligned} \tag{4.2}$$

where $i \in [1, N]$ indexes the supervised training examples $(x_i, \tilde{y}_i)$. The optimal θ^* can be obtained using numerical methods.[3]

4.2 Discriminative Training with Missing Input

4.2.1 Minimum Imputed-Risk

We now turn to the question of how to make use of unsupervised data—specifically, training examples i for which we know only $\tilde{y}_i$ but not x_i. For such i, we cannot compute the summand $\mathrm{L}(\delta_\theta(x_i), \tilde{y}_i)$ in (4.2). Instead we propose to replace $\mathrm{L}(\delta_\theta(x_i), \tilde{y}_i)$ with the

[1] One should not confuse this with the minimum risk training described in Section 2.6.5 on page 34. In both cases, the term *risk* means *expected loss*, but the expectation is taken under different distributions. In particular, the expectation in (2.24) on page 34 is taken under the conditional distribution $p(y \mid x)$, while here the expectation is taken under the joint distribution $p(x, y)$.

[2] In the terminology of statistical decision theory, $p(x, y)$ is a distribution over states of nature. We seek a *decision rule* $\delta_\theta(x)$ that will incur low expected loss on *observations* x that are generated from unseen states of nature.

[3] To compensate for the shortcut of using the unsmoothed empirical distribution rather than a posterior estimate of $p(x, y)$ (Minka, 2000), it is common to add a regularization term $||\theta||_2^2$ in the objective of (4.2). The added regularization term can prevent overfitting to a training set that is not large enough to learn all parameters. A recent alternative regularizes θ more indirectly, by minimizing the empirical risk subject to constraints that ensure that δ_θ not only outputs low-loss translations on the training set, but internally prefers them to high-loss translations by a comfortable margin (Taskar et al., 2005; Crammer et al., 2006).

expectation

$$\left(\sum_x p_\phi(x \mid \tilde{y}_i) \mathrm{L}(\delta_\theta(x), \tilde{y}_i) \right), \tag{4.3}$$

where p_ϕ is a "reverse prediction model" that attempts to impute the missing x_i data. The resulting variant of (4.2), what we called **minimum *imputed* empirical risk** (abbreviated as *minimum imputed-risk*), is[4]

$$\theta^* \quad = \quad \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \sum_x p_\phi(x \mid \tilde{y}_i) \mathrm{L}(\delta_\theta(x), \tilde{y}_i) \tag{4.4}$$

Our minimum imputed-risk objective of (4.4) could be evaluated by *brute force* as follows.

1. For each example $\tilde{y}_i$, use the reverse model p_ϕ to impute its possible reverse translations $\{x_{i1}, x_{i2}, \ldots\}$, and add each $(x_{ij}, \tilde{y}_i)$ pair (weighted by $p_\phi(x_{ij} \mid \tilde{y}_i) \leq 1$) to an imputed training set .

2. Now do ordinary supervised training (as of (4.2)) on the (weighted) imputed training data.

The second step means that we must use δ_θ to forward-translate each imputed x_{ij}, evaluate the loss of the translations against the corresponding true translation $\tilde{y}_i$, and choose the θ that minimizes the weighted sum of these losses (i.e., the empirical risk when the empirical distribution $\tilde{p}(x, y)$ is derived from the imputed training set). Specific to our MT task,[5] this tries to ensure that probabilistic "round-trip" translation (from the target-language sentence to the source language and back again) will have a low (expected) loss.

The trouble is that a typical reverse model p_ϕ will generate a weighted lattice or hypergraph $\mathcal{X}_i$ encoding exponentially many translations of $\tilde{y}_i$. It is computationally infeasible to forward-translate *each* of the $x_{ij} \in \mathcal{X}_i$. We will present several approximations in Section 4.2.4.

4.2.2 The Reverse Prediction Model p_ϕ

The crucial ingredient here is p_ϕ, a "reverse prediction model" that attempts to impute the missing x_i data. We will train this p_ϕ model in advance (before we tune θ), doing the

[4] We can also exploit both supervised and unsupervised data to perform semi-supervised training by using an interpolated version of (4.2) and (4.4). We will do this in our experiments.

[5] Our method may be used for other tasks as well. For example, in a speech recognition task, δ_θ is a speech recognizer that produces text, whereas p_ϕ is a kind of speech synthesizer that must produce a distribution over audio (or at least over acoustic features or phone sequences) (Huang, Li, and Acero, 2010).

best job we can from available data—including our bilingual $(x_i, \tilde{y}_i)$ data as well as any available monolingual x data.[6] Note that ϕ is fixed when we tune θ.

In the MT setting, δ_θ and p_ϕ may have similar parameterization. One translates Chinese to English; the other translates English to Chinese.

Yet the setup is not quite symmetric. Whereas δ_θ is a translation *system* that aims to produce a *single*, *low-loss* translation, the reverse version p_ϕ is rather a probabilistic *model*. It is supposed to give an accurate probability distribution over possible values of the missing input sentence x_i. All of these values will be taken into account in (4.4), without regard to the loss that they would incur if they were evaluated in a reverse MT competition.

Thus, ϕ does not need to be trained discriminatively itself (so there is no circularity). Ideally, it should be trained to match the actual conditional distribution of x given y, by achieving a low conditional cross-entropy, as follows[7]

$$\begin{aligned} \phi^* &= \arg\min_\phi \mathrm{H}_\phi(Y \mid X) \\ &= \arg\min_\phi -\sum_{x,y} p(x,y) \log p_\phi(x \mid y) \end{aligned} \tag{4.5}$$

It may be tolerable for p_ϕ to impute mediocre translations x_{ij}. All that is necessary is that $\delta_\theta(x_{ij})$ resembles the hypotheses in $\mathrm{D}(x_i)$. In other words, the hypergraph generated in the forward translation of the imputed x_{ij} should "simulate" the hypergraph that we would see if we were translating the correct Chinese sentence x_i.

4.2.3 The Forward Translation System δ_θ and the Loss Function $\mathrm{L}(\delta_\theta(x_i), \tilde{y}_i)$

The minimum empirical risk objective of (4.2) is quite general and various popular supervised training methods (Lafferty et al., 2001; Collins, 2002; Och, 2003; Crammer et al., 2006; Smith and Eisner, 2006) can be formalized in this framework by choosing different functions for δ_θ and $\mathrm{L}(\delta_\theta(x_i), \tilde{y}_i)$. The generality of (4.2) extends to our minimum imputed-risk objective of (4.4). Below, we specify the δ_θ and $\mathrm{L}(\delta_\theta(x_i), \tilde{y}_i)$ we considered in our investigation.

Deterministic Decoding

A simple translation rule would define

$$\delta_\theta(x) = \operatorname*{argmax}_y p_\theta(y \mid x) \tag{4.6}$$

[6] In a translation task from x to y, one usually does not make use of monolingual x data. But, in our case, we can use x data to train a language model that will be useful for the reverse system. In particular, it will make the imputed x_i look more like true Chinese inputs.

[7] Empirically, a standard method for this is to minimize $-\frac{1}{N}\sum_{i=1}^{N} \log p_\phi(x_i \mid \tilde{y}_i) + \frac{1}{2\sigma^2}||\phi||_2^2$, where the regularization coefficient σ^2 is tuned on some held out bitext.

where $p_\theta(y \mid x)$ is a log-linear model as defined in (2.18) on page 31. Note that in practice, solving (4.6) is intractable due to spurious ambiguity and we use a Viterbi approximation (see Section 2.5.1 on page 29 for more details). If this $\delta_\theta(x)$ function is used together with a loss function $\mathrm{L}(\delta_\theta(x_i), \tilde{y}_i)$ that is negated BLEU score, our minimum imputed-risk objective of (4.4) is equivalent to MERT (Och, 2003) on the (weighted) imputed training data.[8]

However, this would not yield a differentiable objective function. Infinitesimal changes to θ could result in discrete changes to the winning output string $\delta_\theta(x)$, and hence to the loss $\mathrm{L}(\delta_\theta(x), \tilde{y}_i)$. Och (2003) developed a specialized line search to solve the optimization problem. This is not scalable when δ_θ has a large number of parameters θ.

Randomized Decoding

Instead of using the $\arg\max$ of (4.6), we assume *during training* that $\delta_\theta(x)$ is itself a random quantity: the system *randomly* outputs a translation, choosing y with probability $p_\theta(y \mid x)$. As a result, we will modify our objective function to take yet another expectation over an unknown quantity, replacing $\mathrm{L}(\delta_\theta(x), \tilde{y}_i)$ in (4.4) with

$$\sum_y p_\theta(y \mid x)\ \mathrm{L}(y, \tilde{y}_i) \tag{4.7}$$

Now, our final minimum imputed-risk objective of (4.4) becomes,

$$\theta^* \quad = \quad \operatorname*{argmin}_\theta \frac{1}{N} \sum_{i=1}^{N} \sum_{x,y} p_\phi(x \mid \tilde{y}_i) p_\theta(y \mid x) \mathrm{L}(y, \tilde{y}_i) \tag{4.8}$$

If the loss function $\mathrm{L}(y, \tilde{y}_i)$ is the negated BLEU, this is equivalent to performing minimum-risk training (see Section 2.6.5 on page 34)[9] on the (weighted) imputed data.[10]

The objective function of (4.8) is now differentiable (since each coefficient $p_\theta(y \mid x)$ is a differentiable function of θ), and thus we are able to optimize θ by a gradient-based method. The gradients can be computed by using a second-order expectation semiring (see Section 3.4 on page 50 for details).

4.2.4 Approximating $p_\phi(x \mid \tilde{y}_i)$

As mentioned at the end of Section 4.2.1, it is computationally infeasible to forward-translate *each* of the $x_{ij} \in \mathcal{X}_i$ (imputed by the reverse model p_ϕ), since both the reverse and forward sysmtems we use are Hiero (Chiang, 2007) that uses a SCFG. We suggest three approximations that are computationally more feasible. Each can be regarded as a different approximation of $p_\phi(x \mid \tilde{y}_i)$ in equation (4.4).

[8] One can manipulate the loss function to support other methods (such as Perceptron (Collins, 2002) and MIRA (Crammer et al., 2006)) that use deterministic decoding.

[9] One should not confuse this with the minimum risk training of (4.1).

[10] Again, one may manipulate the loss function to support other probabilistic methods (such as CRF (Lafferty et al., 2001)) that use randomized decoding.

4.2.4.1 k-best

For each $\tilde{y}_i$, add to the imputed training set only the k most probable translations $\{x_{i1}, \ldots x_{ik}\}$ according to $p_\phi(x \mid \tilde{y}_i)$. (These can be extracted from $\mathcal{X}_i$ using standard algorithms (Huang and Chiang, 2005).) Rescale their weights to sum to 1.

4.2.4.2 Sampling

For each $\tilde{y}_i$, add to the training set k independent random samples $\{x_{i1}, \ldots x_{ik}\}$ from the distribution $p_\phi(x \mid \tilde{y}_i)$, each with weight $1/k$. (These can be sampled from $\mathcal{X}_i$ using standard algorithms (Johnson, Griffiths, and Goldwater, 2007).) This method is known in the literature as *multiple imputation* (Rubin, 1987).

4.2.4.3 Lattice

Under certain circumstances it is possible to compute the quantity in equation (4.3) exactly via dynamic programming. Although $\mathcal{X}_i$ does contain exponentially many translations, it uses a "packed" representation in which these translations share structure. Thus, it is sometimes possible to share work and efficiently forward-translate the entire set of imputed translations in $\mathcal{X}_i$, obtaining a distribution over translations y—and then to measure the expected loss under that distribution as required by equation (4.3).

This appears to be possible, by a construction due to Jason Eisner (p.c.), if (a) the posterior distribution $p_\phi(x \mid \tilde{y}_i)$ is represented by an *unambiguous* weighted finite-state automaton $\mathcal{X}_i$, (b) the forward translation system δ_θ is structured in a certain way as a weighted synchronous context-free grammar, and (c) the loss function decomposes in a way that is amenable to dynamic programming.

In our experimental setting described below, (b) is true (using **Joshua**), and (c) is true (since we use a loss function of (2.14) on page 30, which is an approximation to BLEU and decomposable). While (a) is not true in our setting because $\mathcal{X}_i$ is a hypergraph (which is ambiguous), we will show in Chapter 6 on page 89 how to *approximate* a hypergraph representation of $p_\phi(x \mid \tilde{y}_i)$ by an unambiguous WFSA.[11] With this approximation, we can then compute (4.3), and thus find optimal weight vector θ in (4.4).

4.2.4.4 Rule-level Composition

In the previous subsection, we employed a weighted finite-state approximation to the distribution $\mathcal{X}_i$ over imputed Chinese sentences. Typically, however, our reverse translation model p_ϕ will have the same architecture as our forward system p_θ—the Hiero model–and thus will have the form of a weighted synchronous context-free grammar (SCFG). As a result, it will produce a $\mathcal{X}_i$ that is structured like a weighted hypergraph. Just as in the

[11] Note that the forward translation of a WFSA is possible by using a lattice-based decoder (Dyer, Muresan, and Resnik, 2008).

forward system, the hyperedges consider both the English-to-Chinese translation score and the Chinese language model score; this means that they must be specialized according to the Chinese language model state.

Since a hypergraph is a generalization of a finite-state automaton, would it be possible to forward-translate this hypergraph, just as we forward-translated its finite-state approximation in the previous subsection, and thus compute equation (4.3) *exactly*?

Unfortunately not. If the reverse and forward systems were arbitrary SCFGs, then this problem would be a generalization of the intersection of context-free languages, which is known to be undecidable. The situation here is actually not quite that bad—a Hiero system will always produce an *acyclic* hypergraph $\mathcal{X}_i$, so that (4.3) could be computed exactly by individually translating each of the *finitely* though exponentially many $x_i \in \mathcal{X}_i$. Nonetheless, we know of no *efficient* way to compute (4.3) exactly.

It may seem surprising that the structure-sharing in the hypergraph $\mathcal{X}_i$ cannot be exploited as it was in the finite-state automaton of the previous section. Intuitively, the reason is that when the forward Hiero system translates a string $x_i \in \mathcal{X}_i$, it must parse it into recursive phrases. But the structure-sharing within the hypergraph of $\mathcal{X}_i$ has already parsed x_i into recursive phrases, in a way determined by the reverse Hiero system; each phrase correspondings to a hyperedge.

To exploit structure-sharing, we must use a forward translation system that decomposes according to that existing parse of x_i. We can do that by considering *only* forward translations that respect the hypergraph structure of $\mathcal{X}_i$. The simplest way to do this is to require complete isomorphism of the SCFG trees used for the reverse and forward translations.

To implement this approximation, consider one of the hyperedges $\langle e_1, c, L_c \rangle$ in $\mathcal{X}_i$, which represents a possible reverse translation of a possible English phrase e_1 to a Chinese phrase c, in a Chinese context L_c. Here L_c represents a state of the Chinese language model used in the reverse translation; the weight of the hyperedge is sensitive to this. When the forward-translation system wishes to forward-translate any of the imputed Chinese sentences x_i that can be derived with the help of this hyperedge, it is required (under our proposed approximation) to forward-translate c *as a phrase* rather than parsing x_i *in some other way that may not use* c. The construction for forward-translating $\mathcal{X}_i$ under this constraint can simply replace this hyperedge with several hyperedges of the form $\langle e_1, c, L_c, e_2, L_e \rangle$, each of which represents a round-trip phrasal translation from e_1 to c and back to some e_2.

Here L_e represents a state of the English language model used in the forward translation, so we do have a substantial expansion of the number of hyperedges: every hyperedge is annotated with states L_c, L_e from *two* language models. One can, however, approximate further by simplifying or eliminating either the English or the Chinese language model. This reduces the number of states.

Computing equation (4.3) correctly under this approximation requires reweighting the hyperedges using a construction similar to the one mentioned in the previous subsection (Eisner, p.c.). However, it is worth noting that if the English language model is dropped altogether—quite a crude approximation—then it is possible to give

4.3 EM vs. Minimum Imputed-Risk

The notion of imputing missing data is familiar from other settings (Little and Rubin, 1987)—in particular the EM algorithm, which is widely used in NLP. It is therefore instructive to compare our minimum imputed-risk method to that generative approach.

One can train a joint model $p_\theta(x, y)$ by maximizing the log-likelihood of the data,

$$\operatorname*{argmax}_\theta \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(x_i, \tilde{y}_i) \tag{4.9}$$

Where x_i is missing, one replaces $p_\theta(x_i, \tilde{y}_i)$ with just $p_\theta(\tilde{y}_i) = \sum_x p_\theta(x, \tilde{y}_i)$.

A closely related approach would be to impute the missing x_i precisely as we did in Section 4.2.1, obtaining the maximization problem

$$\operatorname*{argmax}_\theta \frac{1}{N} \sum_{i=1}^{N} \sum_x p_\phi(x \mid \tilde{y}_i) \log p_\theta(x, \tilde{y}_i) \tag{4.10}$$

After this maximization (an "M step"), one commonly updates the imputation model p_ϕ to be the conditionalization of the new joint model p_θ (an "E step"), and then repeats the process. (Note that θ here parameterizes a *joint* model, not a *conditional* model as in Section 4.2.1. Also note that the imputation model ϕ is derived from θ and changes along with the change of θ, not a separate and static ϕ as in Section 4.2.1.) This iterative Expectation-Maximization (EM) procedure converges to a local maximum (or other critical point) of the log-likelihood $\sum_{\tilde{y}_i} \log p_\theta(\tilde{y}_i)$.

So why not simply use these maximum-likelihood training procedures for MT? One reason is that they are not discriminative: the loss function is ignored during training. [12]

A second reason is that training good *joint* models is computationally expensive. Contemporary MT makes heavy use of log-linear probability models, which allows the system designer to inject linguistic intuitions or prior knowledge through a careful choice of features. Computing the M-step objective (4.10) in closed form is difficult if p_θ is an arbitrary log-linear model, because the joint probability $p_\theta(x_i, \tilde{y}_i)$ is then defined as a ratio whose denominator Z_θ involves a sum over all possible sentence pairs (x, y) of any length.

[12] While we could still exploit the loss function at *test* time, by defining our translation system via the minimum Bayes risk (MBR) decoding (as discussed in Section 2.5.2 on page 30),

$$\delta_\theta(x) = \arg\min_y \sum_{y'} \mathrm{L}(y, y') p_\theta(y' \mid x) \tag{4.11}$$

this would make translation computationally expensive at test time: although (4.11) is tractable (for some loss functions) by dynamic programming, it is still more complex than a Viterbi decoder and so cannot exploit A* or coarse-to-fine decoding techniques. More importantly, under (4.11), the system would still not be tuned to minimize its *actual* error. (While the rule (4.11) attempts to *predict* the error of the output y, it may do so inaccurately if p_θ is a poor model, and there is no opportunity to adjust θ to compensate for such modeling flaws.) For this reason, we think it is important to consider loss at training time.

By contrast, our discriminative framework will only require us to work with conditional models. While conditional probabilities such as $p_\phi(x \mid y)$ and $p_\theta(y \mid x)$ are also ratios, computing their denominators only requires us to sum over a packed forest of possible translations of a *given* y or x. Analogously, discriminative CRFs have become more popular than generative HMMs because they permit efficient training even with a wide variety of log-linear features (Lafferty et al., 2001).

In summary, EM would impute missing data using $p_\theta(x \mid y)$ and predict outputs using $p_\theta(y \mid x)$, both being conditionalizations of the same joint model $p_\theta(x, y)$. Our minimum imputed-risk training method is similar, but it instead uses a pair of separately parameterized, separately trained models $p_\phi(x \mid y)$ and $p_\theta(y \mid x)$. By sticking to conditional models, we can efficiently use more sophisticated model features, and we can incorporate the loss function when we train θ, which should improve both efficiency and accuracy at test time.

4.4 Experimental Results

We report results on a Chinese-to-English translation task using **Joshua** (Li et al., 2009a) for both the forward and reverse translation.

4.4.1 Baseline Systems

We train both reverse and forward baseline systems. The translation models are built on a corpus for IWSLT 2005 Chinese to English translation task (Eck and Hori, 2005), which consists of 40k pairs of transcribed utterances in the travel domain.[13] We use a *5-gram* language model with modified Kneser-Ney smoothing (Chen and Goodman, 1998), trained on the English (or Chinese) side of the bitext. We use the standard pipeline (see Section 1.1 on page 1) and pruning settings (Chiang, 2007).

4.4.2 Feature Functions

We use two classes of features for *discriminative training* (cf. Figure 1.1 on page 2).

Regular Hiero Features

We include ten features that are standard in Hiero (Chiang, 2007). In particular, these include one baseline language model feature, three baseline translation models, one word penalty feature, three features to count how many rules with an arity of zero/one/two are used in a derivation, and two features to count how many times the unary and binary glue rules (see page 4 in Section 1.1.1) are used in a derivation.

[13] This task is relatively small, compared with a more realistic task (e.g., the NIST one). We use such a task for computational efficiency.

Target-rule Bi-gram Features

For each bilingual rule, we extract bi-gram features over the target-side symbols (including non-terminals and terminals). For example, if a bilingual rule's target side is "*on the* X_1 *issue of* X_2" where X_1 and X_2 are non-terminals (with a position index), we can extract bi-gram features including: "*on the*", "*the* X_1", "X_1 *issue*", "*issue of*", and "*of* X_2". We consider only those terminal symbols (i.e., regular English words) that occur frequently in the English text. Moreover, for the terminal symbols, we will use their dominant POS tags (instead of the symbol itself). For example, "*on the*" becomes"*prep det*". We use 541 such bi-gram features.

4.4.3 Data Sets for Discriminative Training

We use three bilingual data sets (see Table 4.1): one to train the inverse model p_ϕ[14] (which uses only the 10 standard Hiero features as described in Section 4.4.2),[15] one to train the forward model δ_θ (which uses both classes of features described in Section 4.4.2, i.e., 551 features in total), and one for test.

In all three data sets, each Chinese sentence x_i has 16 English reference sentences, so the gold-standard translation $\tilde{y}_i$ is actually a set of 16 translations. The loss $\mathrm{L}(y, \tilde{y}_i)$ is defined as the negated BLEU metric, which is able to evaluate a single output sentence y against this set $\{\tilde{y}_i\}$.

When we impute data from $\tilde{y}_i$, we impute a one-best translation x_i for each $y \in \{\tilde{y}_i\}$. This effectively adds 16 pairs of the form $(x_i, \tilde{y}_i)$ to the training set (see Section 4.2.4), where each x_i is a different input sentence in each case, but $\tilde{y}_i$ is always the original set of 16 references.[16]

4.4.4 Semi-Supervised Training

Table 4.2 shows the main results of this chapter, which show how our method performs in a semi-supervised training scenario (which is the most likely setting in practice).

We compare three discriminatively trained systems. The supervised system ("Sup") carries out discriminative training on an in-domain bilingual data set having 200 Chinese sentences and 200*16 English references. The other two are semi-supervised: they *also* use some unsupervised in-domain data (i.e., monolingual English) during discriminative training.

[14] In our experiments, for simplicity, we use a dedicated data set for training the reverse model. But in practice, this may not be what someone would really do, especially when only a small amount of bilingual data is available. Instead, one may use this precious data across training both the reverse and forward models.

[15] In the present results, we trained ϕ discriminatively to minimize risk, for implementation convenience. In future work, we will instead train ϕ to minimize the conditional cross-entropy (4.5) as recommended by Section 4.2.2.

[16] In practice, one may not have multiple references. We take the unrealistic advantage of our data set (which has 16 references) for stability of the minimum risk training.

Data set	Purpose	# of sentences	
		Chinese	English
Set1	training ϕ	503	503*16
Set2	training θ	503	503*16
Set3	testing	506	506*16

Table 4.1: **Three data sets for discriminative training.** Note, related to Figure 1.1 on page 2, that additional bitext (40k sentence pairs) are used to train the (generative) translation model and language model.

Training scenario	Test BLEU
Sup, (200, 200*16)	47.6
+Unsup, 101*16 Eng sentences	49.0
+Unsup, 202*16 Eng sentences	48.9
+Unsup, 303*16 Eng sentences	49.7

Table 4.2: **BLEU scores for semi-supervised training.** The supervised system ("Sup") is trained on a bilingual data set having 200 Chinese sentences and 200*16 English references. "+Unsup" means that we add unsupervised data (i.e., monolingual English sentences) for training. For each English sentence, we impute a one-best Chinese translation using the reverse translation system.

Clearly, adding unsupervised data improves over the supervised case, by at least 1.3 BLEU points. However, it is not necessarily true that more unsupervised data is always better.

4.4.5 Supervised and Unsupervised Training

Though the semi-supervised training scenario is the most likely scenario in practice, we are interested in knowing how our unsupervised discriminative method performs when compared with supervised discriminative training, under different training data sizes. Table 4.3 shows relevant results. Surprisingly, our unsupervised method actually performs better than the supervised training in most cases. Also, using more unsupervised data is better in general.

A possible explanation for the high performance of the unsupervised method is that the training set is considerably larger and more diverse in this case. Whereas the supervised method learns to translate only a single observed x_i sentence into something resembling the $\tilde{y}_i$ set, the unsupervised method learns to translate each of 16 imputed x_i sentences back into something resembling the $\tilde{y}_i$ set (as explained above). These imputed x_i sentences can use different vocabulary and sentence structure derived from the different references.

Data size	Test BLEU	
	Sup	Unsup
101	48.6	48.5
202	48.2	48.9
303	47.7	48.8
403	48.2	49.6
503	48.6	49.2

Table 4.3: **BLEU scores for supervised and unsupervised training.** For supervised training ("Sup"), a data size of 101 means that we use a bilingual data set having 101 Chinese sentences and 101*16 English references. For unsupervised training ("Unsup"), a data size of 101 means that we use an English monolingual data set having 101*16 sentences, and for each English sentence we will impute a one-best Chinese translation by using the reverse translation system.

4.4.6 Unsupervised Training with Different Reverse Models

A critical component of our unsupervised method is the reverse translation model $p_\phi(x \mid \tilde{y}_i)$. We wonder how the performance of our unsupervised method changes when the quality of the reverse system varies. To answer this question, we used two different reverse translation systems, one with a language model trained on the Chinese side of the bitext ("WLM"), and the other one without using such a Chinese LM ("NLM"). It is well known that using a language model is critical to the translation performance. This is evident from the BLEU scores of the imputed Chinese translation in Table 4.4. Note that the BLEU scores for the imputed Chinese are quite low, because only one Chinese reference is available for scoring. Clearly, to make our unsupervised method work, we need to have a reasonably good reverse translation system.

4.4.7 Unsupervised Training with Different k-best Sizes

In all the experiments so far, we used the reverse translation system to impute only a single Chinese translation for each English monolingual sentence. (This is the k-best approximation of section 4.2.4 with $k = 1$.)

Table 4.5 shows (in the fully unsupervised case) that the performance does not change much as we increase k.[17] However, even the 5-best sentences are likely to be quite similar to one another (May and Knight, 2006). Imputing a longer k-best list, a sample, or a lattice for x_i (see Section 4.2.4) might achieve more diversity in the training inputs, which might make the system more robust, just as diversity apparently helped performance in section 4.4.5.

[17] In the present experiments, however, we simply weighted all k imputed translations equally, rather than in proportion to their posterior probabilities as in section 4.2.4.

Data size	Imputed-ZH BLEU		Test BLEU	
	WLM	NLM	WLM	NLM
101	11.8	3.0	48.5	46.7
202	11.7	3.2	48.9	47.6
303	13.4	3.5	48.8	47.9

Table 4.4: **BLEU scores for unsupervised training with/without using a language model in the reverse system.** "WLM" means a Chinese language model is used in the reverse system, while "NLM" means no Chinese language model is used. A data size of 101 means that we use an English monolingual data set having 101*16 sentences, and for each English sentence we will impute a one-best Chinese translation by using the reverse translation system. In addition to reporting the test BLEU (of the English translations) as usual, we also report the "Imputed-CN BLEU", which is the BLEU score of the imputed Chinese sentence computed using the true Chinese sentence as a reference.

Training scenario	Test BLEU
Unsup, k=1	48.5
Unsup, k=2	48.4
Unsup, k=3	48.9
Unsup, k=4	48.5
Unsup, k=5	48.4

Table 4.5: **BLEU scores for unsupervised training with different k-best sizes.** We use 101*16 monolingual English sentences, and for each English sentence we impute k-best of Chinese translations using the reverse system.

4.4.8 Goodness of the Simulated Neighborhood

In supervised training, for a given Chinese input sentence, the forward system will generate many English sentences, among which the training will learn to discriminate. This set of English sentences can be thought as a neighborhood of the English reference translation for the given Chinese input. In the unsupervised case, the Chinese input is missing, and so the minimum imputed-risk training imputes the missing Chinese input for each observed English sentence and then forward-translates the imputed Chinese input back to many English sentences. This can be thought as a *simulated* neighborhood of the original English sentence. For the minimum imputed-risk training to work well, we expect that the simulated neighborhood will have some overlap with the true neighborhood that is generated from the true Chinese input.

To measure the goodness of the simulated neighborhood, we obtain the sets of n-gram types in the two neighborhoods, and then compute the ratio between the number of n-gram

n-gram	Precision	Recall
unigram	43.2%	41.2%
bigram	15.2%	14.3%
trigram	6.8%	5.9%
4-gram	3.7%	2.8%

Table 4.6: Precisions and recalls of simulated neighborhood's n-grams. The n-grams are collected from k-best strings where $k = 100$, which corresponds to the first data point in Figure 4.1.

types in the intersection and that in the union. Figure 4.1 shows the results when comparing different sizes of k-best English strings in the neighborhoods. In general, the ratio decreases when n is larger. Also, the ratio does not change much along with the change of k.

Table 4.6 presents the simulated neighborhood's precisions and recalls of the n-grams in the true neighborhood. The n-grams are collected from k-best strings where $k = 100$, which corresponds to the first data point in Figure 4.1 We would conclude that the simulated neighborhood does a reasonably well job in simulating the true neighborhood.

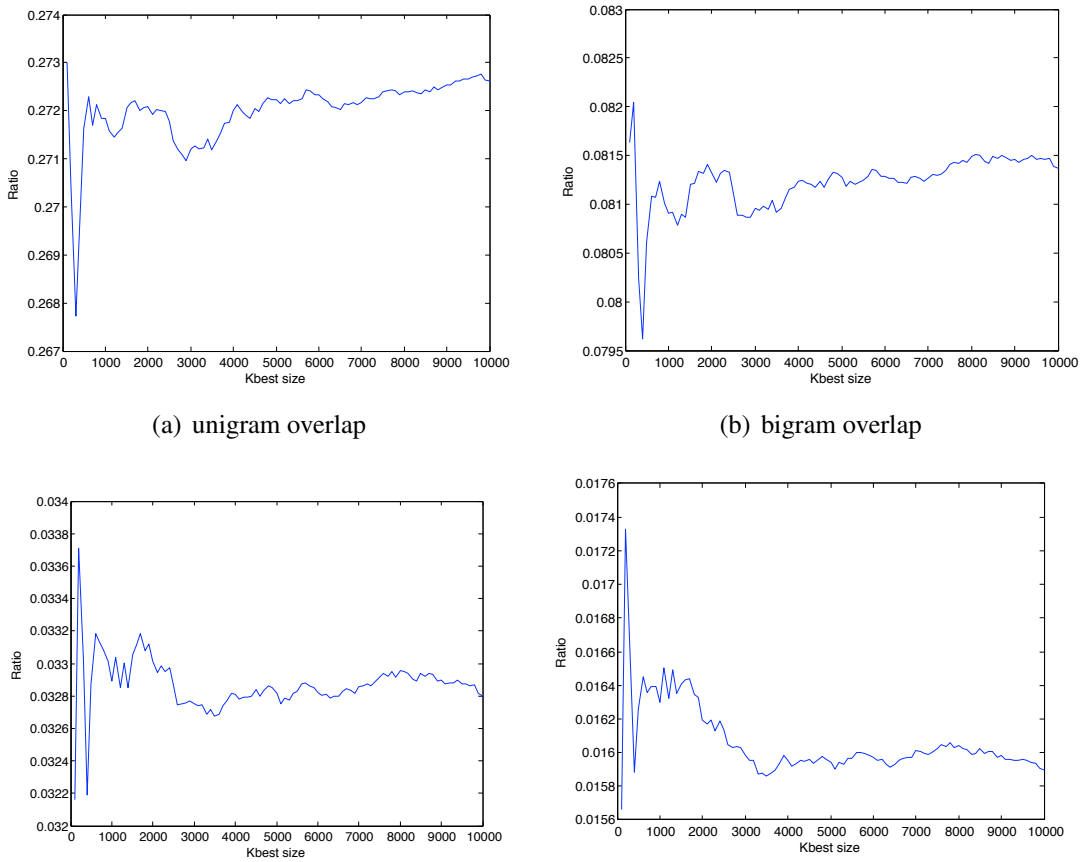
4.4.9 Some Translation Examples

Table 4.7 presents several examples of translations which a semi-supervised system (i.e., the system of the last row in Table 4.2) performs better than a supervised one (i.e., the system of the first row in Table 4.2).

4.5 Summary

In this chapter, we have presented an unsupervised discriminative training method that works with missing inputs. Our method first uses a reverse model to impute the missing input, and then optimizes the parameters of the forward system such that the imputed risk is minimized.

We have applied our method to a Chinese to English machine translation task. Supervised data are used to train baseline models, which are then combined with one another and with additional features by discriminative training. We show that unsupervised training in the discriminative phase performs as well as supervised training, and often better. Furthermore, augmenting supervised data with unsupervised data improves performance.



(a) unigram overlap

(b) bigram overlap

(c) trigram overlap

(d) 4-gram overlap

Figure 4.1: **The goodness of the simulated neighborhood by minimum imputed-risk.** We use the neighborhoods generated by the last two systems (supervised and unsupervised) in Table 4.3. In the unsupervised case, for each test English sentence, we impute a one-best Chinese, and then use the trained forward system to translate it back to English. For each true (or imputed) Chinese, we generate a k-best English strings. We then obtain two sets of n-gram types (that occur in the two k-best strings), and compute the ratio between the number of n-gram types in the intersection and that in the union. Finally, we obtain the average ratio among different sentences in the test set (i.e., *set3*).

System	Output
Chinese	我们 应该 什么 时候 来 ?
Reference	what time shall we come ?
Supervised	when should we ?
Semi-supervised	what time should we come ?
Chinese	我 要 吃 鱼 。
Reference	i 'll have the fish .
Supervised	i like fish .
Semi-supervised	i 'd like to have fish .
Chinese	预约 柜台 在 哪 ?
Reference	where is the reservation counter ?
Supervised	where reservation counter ?
Semi-supervised	where is the reservation counter ?
Chinese	我 想试 一下 本 地 葡萄酒
Reference	i 'd like to try some local wine .
Supervised	i try local wine .
Semi-supervised	i 'd like to try a local wine .
Chinese	我 房间 的 热水 没 了 。
Reference	the hot water does n't work in my room .
Supervised	my room nothing hot water .
Semi-supervised	the hot water is not in my room .
Chinese	哪 天 和 我 一起 吃午饭 怎么 样 ? 我 在 我 的 旅馆 附近 发现 了 一 家 好餐馆 。
Reference	how about having lunch with me some day ? i found a good restaurant near my hotel .
Supervised	what day how lunch with me ? i 'm my hotel found a restaurant around here .
Semi-supervised	how about lunch with me what day ? i found a good restaurant near at my hotel .

Table 4.7: Examples of translation outputs which a semi-supervised system (i.e., the system of the last row in Table 4.2) performs better than a supervised one (i.e., the system of the first row in Table 4.2).

Chapter 5

Contrastive Language Model Estimation

The minimum imputed-risk training presented in the previous chapter requires to use two full-scale MT systems: the backward and the forward systems. In this chapter, we propose another unsupervised method, *contrastive language model estimation*, which can also exploit monolingual English data to perform discriminative training, but does not require a reverse system. The method proposed in this chapter can be thought as an approximation to the intractable minimum imputed-risk objective of (4.8) on page 60. In particular, it resembles the *rule-level composition* approximation in Section 4.2.4.4 on page 61 (for a detailed comparison, see Section 5.3.4).

Our method works as follows. We first extract a *confusion grammar* from a *bilingual grammar*. Specifically, whenever in the bilingual grammar we see two rules that have the same Chinese side (say c) but two different English sides (say e_1 and e_2), we will extract a confusion rule $X \rightarrow \langle e_1, e_2 \rangle$. The confusion rule is English to English and captures the confusion that an MT system may have when translating the Chinese-side c. Now, given a good English sentence $\tilde{y}$, we use the confusion grammar to produce many alternative English sentences y. This can be done as regular MT decoding, as the confusion grammar can be treated as a "translation" model in an English-to-English "translation" system. The set of y thus generated can be thought as alternative translations generated by the SMT system if it had known the corresponding Chinese input x. Now, we can train a discriminative model on the generated data y (with the original English sentence $\tilde{y}$ as the training reference) such that the original sentence $\tilde{y}$ will be highly preferred by the model. The model trained can then be used as a regular language model for actual MT decoding (e.g., translating Chinese to English).

We again perform experiments by using the open-source MT toolkit **Joshua** (Li et al., 2009a). We show that the contrastive language model (CLM) performs better than a regular n-gram LM in terms of recovering an English sentence from its neighborhood (i.e., a set of alternative sentences that are generated from the English sentence). The CLM also improves the performance of an MT system.

5.1 Unsupervised Training of Global Log-Linear Language Models

We have a set of training examples $\tilde{y}_i$, where each $\tilde{y}_i$ is a sequence of English words. We aim to train a language model $p_\theta(y)$ (parameterized by θ) over the examples. The model will be used to assign a probability to any English sentence. We can obtain such a model by maximizing the likelihood of the training examples as follows,

$$\theta^* = \arg\max_\theta \prod_i p_\theta(\tilde{y}_i) \tag{5.1}$$

Now, the question is: what is the form of $p_\theta(\tilde{y}_i)$? We can specify p_θ as a globally normalized log-linear model as follows,

$$p_\theta(y) = \frac{e^{f(y)\cdot\theta}}{\mathrm{Z}(*)} \tag{5.2}$$

where $f(y)$ is a feature vector depending on y, θ is the corresponding weight vector, and $\mathrm{Z}(*) \stackrel{\text{def}}{=} \sum_{y'\in\Sigma^*} e^{f(y')\cdot\theta}$ is a normalization constant. This is called as a whole-sentence maximum entropy language model (Rosenfeld, Chen, and Zhu, 2001).[1]

Training with the above log-linear model requires computing the normalization constant $\mathrm{Z}(*)$, which is computationally challenging as it requires to sum over $y \in \Sigma^*$ (which is the set of all possible English sentences with any length!). To address the computational difficulty issue, Rosenfeld et al. (2001) approximately compute $\mathrm{Z}(*)$ using a set of sentences sampled randomly from Σ^*.

In addition to the computational disadvantage, the model above also has modeling limitation. In particular, in a task like MT, the point of an LM is to discriminate between $\tilde{y}$ from a *confusion set*, which contains alternative translations considered by the MT system for a given Chinese input. In the model above, the confusion set is Σ^*, which is unrealistic for an MT system. A better way to train an LM is to use a real confusion set that is generated by an MT system. However, this requires bilingual training data. Therefore, we propose the following model,

$$p_\theta(\tilde{y} \mid \mathcal{N}(\tilde{y})) = \frac{e^{f(\tilde{y})\cdot\theta}}{\mathrm{Z}(\tilde{y})} \tag{5.3}$$

$$= \frac{e^{f(\tilde{y})\cdot\theta}}{\sum_{y'\in\mathcal{N}(\tilde{y})} e^{f(y')\cdot\theta}} \tag{5.4}$$

where $\mathcal{N}(\tilde{y})$ is a *simulated* confusion set of $\tilde{y}$ and can be obtained by applying a confusion grammar (see Section 5.2) on $\tilde{y}$. Our hope is that the simulated confusion set resembles

[1] One should not confuse this with a regular maximum entropy language model of (5.9), where the normalization is done for each n-gram history (i.e., locally normalized).

the actual confusion set an MT system will generate if it were given the Chinese input corresponding to $\tilde{y}$. Our method is quite similar to the *contrastive estimation* (CE) by Smith and Eisner (2005), but which several important differences (see Section 5.3.1).

5.2 Contrastive Language Model Estimation for MT

Training a contrastive language model for MT involves in the following steps.

- First, extract a *confusion grammar* (CG), which is an English-to-English grammar and captures the confusion an MT system might have when choosing different translation options for a given Chinese input.
- Then, for each English sentence in our monolingual corpus, use the confusion grammar (as a "translation" model) to generate a *neighborhood* (i.e., many alternative English sentences).
- Finally, train a contrastive language model on the neighborhoods (with their corresponding original English sentences as references) by using a discriminative training method.

The trained model can then be used for actual MT decoding. Below, we present details for each step.

5.2.1 Extracting a Confusion Grammar

We will first define the form of the confusion grammar and then describe ways in extracting such a grammar.

Confusion Grammar: Monolingual SCFG

We assume a formalism of synchronous context free grammar (SCFG) for the confusion grammar (CG). While a typical SCFG is bilingual, our CG is *monolingual* since both the source and target sides are English. Some example rules in the CG are as following,

$$X \rightarrow \langle \text{ a cat , the cat } \rangle,$$

$$X \rightarrow \langle X_0 \text{ at beijing , beijing 's } X_0 \rangle,$$
$$X \rightarrow \langle X_0 \text{ of } X_1 \text{ , } X_0 \text{ of the } X_1 \rangle,$$
$$X \rightarrow \langle X_0 \text{ 's } X_1 \text{ , } X_1 \text{ of } X_0 \rangle.$$

Like a regular SCFG, a CG contains rules with different arities. Also, there might be reordering in the rule as shown in the last example. These rules captures the confusion that an MT system may have in choosing different *senses* or *reordering patterns* for a given input. Now the question is how to acquire such a grammar. Below we present two ways.

Extracting Confusion Grammar from Bilingual Grammar

We can derive a confusion grammar from an existing bilingual grammar. For a particular Chinese side (say c), a bilingual grammar may have many different English translation options (say $e \in E$). (Note that $e \in E$ may contain both terminals and nonterminals.) For each pair of such translation options (say $e_1 \in E$ and $e_2 \in E$), we can extract two confusion rules: $X \rightarrow \langle e_1, e_2 \rangle$ and $X \rightarrow \langle e_2, e_1 \rangle$. These rules capture the confusion that an MT system will have when translating the Chinese-side c. Clearly, we can extract $|E|^2$ such confusion rules given a set of translation options E (for the Chinese side c). Note that for each English side e in the bilingual grammar, we also extract an identity rule, that is $X \rightarrow \langle e, e \rangle$. Also note that the rules in the CG are unweighted.

In practice, the above extraction process can be very efficient. Specifically, we read the bilingual grammar into a Trie data structure where the prefixes correspond to the Chinese sides of the bilingual grammar, and thus the rules at each node in the Trie will be the different translation options for the Chinese side (i.e., the prefix at the node). In this way, we can quickly identify the different translation options for the same Chinese side, and thus extract the confusion grammar efficiently.

Extracting Test-set Specific Confusion Grammar

In the procedure described above, the bilingual grammar contains all the rules that are extractable from the bilingual training corpora, and thus the confusion grammar derived from it might be very big. A standard trick is to use a test-set specific grammar. That is, we can first extract a bilingual grammar that is specific to the test set, and then derive a confusion grammar from the bilingual grammar.

Even further, one may extract a CG from the hypergraphs that are generated for the test-set.[2] Recall that a node in a hypergraph corresponds to the same source span and thus the same Chinese in the input, and that a node has many incoming hyperedges each of which is associated with a bilingual rule. Clearly, all the bilingual rules associated with the incoming hyperedges of a given node will have the same Chinese side. Therefore, at each node, we will extract confusion rules for those English sides (which are the target sides of the bilingual rules associated with the node's incoming hyperedges) that have the same arity.

The CG extracted from the test-set hypergraphs might be different from the one derived directly from the test bilingual grammar. This is true since we often perform pruning when generating the hypergraphs and the pruning might be guided by some other models (e.g., a regular n-gram language model) in addition to the bilingual grammar. Such pruning might already remove much confusion that exists in the bilingual grammar but is resolved by other models during the actual MT decoding. Therefore, the CG extracted from the test-set hypergraphs will tend to be smaller and may be more precise in term of capturing confusion in actual MT decoding.

[2]Note that we do not need to look at the English reference translations of the test-set during the extraction.

5.2.2 Generating Simulated Neighborhood

Now, given a good English sentence $\tilde{y}$, we use the confusion grammar to produce many alternative English sentences (i.e., a neighborhood $\mathcal{N}(\tilde{y})$). This can be done as regular MT decoding as we can treat the CG as a regular "translation" model in an English-to-English "translation" system.

To make sure that we produce at least one derivation for each $\tilde{y}$. We first need to add into CG the standard glue rules as in Hiero (Chiang, 2007).

$$S \rightarrow \langle X_0 , X_0 \rangle ,$$

$$S \rightarrow \langle S_0 X_1 , S_0 X_1 \rangle ,$$

We also need to add an out of vocabulary (OOV) rule $X \rightarrow \langle word, oov \rangle$ for each *word* in $\tilde{y}$ and set the cost of such rule at a maximum value such that the OOV rule will get used only when the CG does not know how to "translate" the *word*. Note that the CG does contain identity rules for English words and phrases that occur in the bilingual grammar. But, this does not guaranty that the rules in CG alone are able to cover all the words in $\tilde{y}$ since some English words may not be in the English vocabulary of the CG. Therefore, we may need to use the OOV rules. [3]

Since the CG is an SCFG, the neighborhood $\mathcal{N}(\tilde{y})$ generated for the observed $\tilde{y}$ is a hypergraph, encoding not only the alternative sentences of $\tilde{y}$ but also the hierarchical process (e.g., which phrase in $\tilde{y}$ has been replaced with what) about how the alternative sentences are generated. The hierarchical process is represented in a derivation tree. As usual, many different derivation trees may correspond to the same string/sentence due to spurious ambiguity. We use $\mathrm{D}(\tilde{y})$ to represent the set of derivations in the hypergraph (that is generated for $\tilde{y}$). The set of generated y can be thought as a simulation of the alternative translations that would have been generated by the MT system if it had known the corresponding Chinese input x for $\tilde{y}$. Figure 5.1 presents an example of hypergraph, which contains four alternative strings *the cat the mat*, *the cat 's the mat*, *the mat of the cat*, and *the mat on the cat* that are generated by CG for the English sentence *a cat on the mat*.

[3] Note that the use of the glue and OOV rules together with the test-set specific confusion grammar (see Section 5.2.1), our SCFG parsing may lead to degenerate behavior. Imagine we have the following English sentence,

John walked home by a different route that evening.

If our test-set specific confusion grammar only has confusion rules $X \rightarrow \langle$walked home, drove homeward$\rangle$ and $X \rightarrow \langle$a, the$\rangle$. The confusion set that the parsing produces will include,

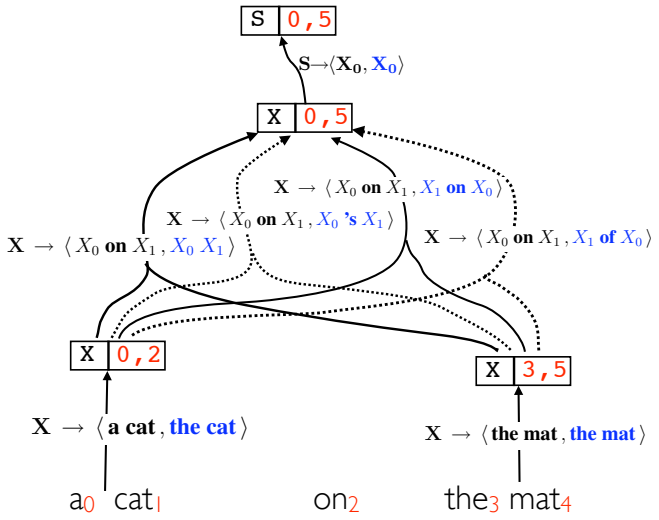
oov [walked home] oov [a] oov oov oov oov

oov [drove homeward] oov [the] oov oov oov oov

But training the contrastive LM to discriminate between the examples above will get almost identical results to simply counting the number of phrasal matches on the English without any parsing. So, parsing might be a waste of time in this case (although parsing might be as fast as pattern matching of the phrases in English in this case).

$$
\begin{array}{l}
X \rightarrow \langle \text{a cat} , \text{ the cat} \rangle \\
X \rightarrow \langle \text{the mat} , \text{ the mat} \rangle \\
X \rightarrow \langle X_0 \text{ on } X_1 , X_0 \ X_1 \rangle \\
X \rightarrow \langle X_0 \text{ on } X_1 , X_0 \text{ 's } X_1 \rangle \\
X \rightarrow \langle X_0 \text{ on } X_1 , X_1 \text{ on } X_0 \rangle \\
X \rightarrow \langle X_0 \text{ on } X_1 , X_1 \text{ of } X_0 \rangle \\
S \rightarrow \langle X_0 , X_0 \rangle
\end{array}
$$

(a) An example confusion grammar.



(b) An example hypergraph generated by the confusion grammar of (a).

Figure 5.1: **Confusion grammar and an example hypergraph generated by the confusion grammar.** Given an input sentence "a cat on the mat", the confusion grammar of (a) may generate a hypergraph (i.e., a neighborhood) for the input sentence, where the hypergraph contains four alternative sentences "the cat the mat", "the cat 's the mat", "the mat on the cat", and "the mat of the cat".

5.2.3 Discriminative Training

With the observed sentence $\tilde{y}$ and its neighborhood $\mathcal{N}(\tilde{y})$ (and $\mathrm{D}(\tilde{y})$), we can then perform the regular discriminative training. We use the minimum risk training (see also

Section 2.6.5 on page 34)[4] that is,

$$\begin{aligned}\theta^* &= \arg\min_\theta \sum_i \text{Risk}_\theta(\tilde{y}_i) \\ &= \arg\min_\theta \sum_i \sum_{y \in \mathcal{N}(\tilde{y}_i)} \text{L}(y, \tilde{y}_i) p_\theta(y \mid \mathcal{N}(\tilde{y}_i)) \\ &= \arg\min_\theta \sum_i \sum_{d \in \text{D}(\tilde{y}_i)} \text{L}(\text{Y}(d), \tilde{y}_i) p_\theta(d \mid \text{D}(\tilde{y}_i)),\end{aligned} \tag{5.5}$$

where $p_\theta(d \mid \tilde{y}_i)$ is defined as,

$$p_\theta(d \mid \text{D}(\tilde{y}_i)) = \frac{e^{f(d)\cdot\theta}}{\sum_{d \in \text{D}(\tilde{y}_i)} e^{f(d)\cdot\theta}} \tag{5.6}$$

where θ is the contrastive model we aim to train, and $f(d)$ is a feature vector over d, which we will specify in Section 5.4. In general, the feature functions should be defined in a way such that the training will be efficient and the actual MT decoding can use them conveniently.

The objective of (5.5) is differentiable and thus we can optimize θ by a gradient-based method. The risk and its gradient can be computed by using a second-order expectation semiring (see Section 3.4 on page 50 for details).

In practice, the full contrastive set $\mathcal{N}(\tilde{y})$ defined by a confusion grammar (CG) may be too large and we have to perform pruning during training. But, the pruning itself may depend on the contrastive model that we aim to train. How do we solve this circular dependency problem? We adopt the following procedure. Given an initial contrastive model θ, we generate a hypergraph (with pruning) for each $\tilde{y}$, and train an optimal θ^* of (5.5) on these hypergraphs. Then, we use the optimal θ^* to *regenerate* a hypergraph for each $\tilde{y}$, and do the training again. This iterates until convergence. This procedure is quite similar to the k-best MERT (Och, 2003) where the training involves a few iterations and each iteration uses a new k-best list that is generated using the latest model.

5.2.4 Applying The Contrastive Language Model

First, we can measure the goodness of the CLM in a simulated task to see how well it can recover $\tilde{y}$ from its neighborhood $\text{D}(\tilde{y})$. This is merely a proof of concept, and may be useful in deciding which features to employ for discriminative training.

The intended use of the CLM is, of course, for actual MT decoding (e.g., translating Chinese to English). Specifically, we can add the contrastive model into an MT pipeline, and tune its weight relative to other existing models in the MT system.

[4] One can also use other discriminative training methods described in Section 2.6.

5.3 Comparison to Related Work

5.3.1 Comparison to CE

Our method is similar to the contrastive estimation (CE) (Smith and Eisner, 2005). In particular, our confusion grammar is like a neighborhood function in CE. Also, our goal is to improve both efficiency and accuracy, just as CE does. However, there are several important differences. First, the setup is quite different. While CE is originally proposed for unsupervised prediction of x from y (e.g., predicting a part of speech (POS) sequence from an observed English sentence),[5] our goal is to purely train a language model (i.e., $p(y)$) on y. (That is, we do not aim to predict a x from y.) Secondly, the neighborhood function in CE is manually created and independent from any particular task, while our neighborhood function (i.e., the confusion grammar) is automatically learnt (e.g., from the bilingual grammar) and is specific to our MT task. Therefore, our neighborhood function might be more informative and adaptive to a given task. Thirdly, when tuning θ, CE uses the maximum likelihood training of (5.1), but we use the minimum risk training of (5.5). Since our training uses a task-specific loss function, it is expected to perform better than the maximum likelihood training.

5.3.2 Locally Normalized Language Model

Since our contrastive model can be used as a language model $p(y)$, it is instructive to compare our method to other language modeling techniques. The most commonly used LM is the so-called n-gram model parameterized as follows,

$$p(y) \quad = \quad \prod_{w \in W_n} p(r(w) \mid h(w))^{c_w(y)} \tag{5.7}$$

where W_n is a set of n-gram types. Each $w \in W_n$ is an n-gram, which occurs $c_w(y)$ times in the string y, and w may be divided into an $(n-1)$-gram prefix $h(w)$ (the *history*) and a uni-gram suffix $r(w)$ (the *rightmost* or *current* word). For example, for a trigram "*on the table*", the history is "*on the*" and the current word is "*table*".

In the n-gram model of (5.7), $p(r(w) \mid h(w))$ is a proper probability distribution normalized by the history $h(w)$, and thus the model is *locally normalized*, while our contrastive language model is globally normalized.

What are the parameters θ in the model? One can treat $p(r(w) \mid h(w))$ directly as the parameters that we aim to learn. A usual way to estimate such parameters is to use a maximum likelihood estimation, that is,

$$p(r(w) \mid h(w)) \quad = \quad \frac{c(w)}{c(h(w))} \tag{5.8}$$

[5] Note that our use of x and y is different from that by Smith and Eisner (2005) (i.e., our x is their y, and vice versa).

where $c(\cdot)$ is the occurrence frequency in the training corpora. This model may assign a zero-probability to a sentence that contains unseen n-gram(s). Thus, the art of n-gram language modeling is about how to smooth the distribution (Chen and Goodman, 1998).

Another choice of parameters θ is to define a log-linear model for $p(r(w) \mid h(w))$ as follows,

$$\begin{aligned} p_\theta(r(w) \mid h(w)) &= \frac{e^{f(h(w),r(w))\cdot\theta}}{\mathrm{Z}(h(w))} \\ &= \frac{e^{f(h(w),r(w))\cdot\theta}}{\sum_{r' \in \Sigma} e^{f(h(w),r')\cdot\theta}} \end{aligned} \tag{5.9}$$

where $f(\cdot,\cdot)$ is a feature vector depending on the history and the current word, and Σ is the vocabulary of words. This corresponds to a maximum entropy language model (Rosenfeld, 1996; Khudanpur and Wu, 2000). Note that the model here is still locally normalized (by the n-gram history), and one should not confuse this with the whole-sentence maximum entropy language model of (5.2).

5.3.3 Globally Normalized Language Model

While our contrastive language model is a global log-linear model trained in an unsupervised way, one can also train such a model in a supervised way (Roark, Saraclar, Collins, and Johnson, 2004; Li and Khudanpur, 2008b). To recall, in supervised training, we know the corresponding x_i for each $\tilde{y}_i$, and we can maximize the conditional likelihood as follows,

$$\theta^* = \arg\max_\theta \prod_i p_\theta(\tilde{y}_i \mid x_i) \tag{5.10}$$

$$= \arg\max_\theta \prod_i \frac{e^{f(x_i,\tilde{y}_i)\cdot\theta}}{\mathrm{Z}(x_i)} \tag{5.11}$$

where the normalization constant $\mathrm{Z}(x_i) = \sum_y e^{f(x_i,y)\cdot\theta}$ requires to sum over *only* the outputs y for the given input x_i. In MT, this corresponds to different translation outputs $y \in \mathrm{T}(x_i)$ for a Chinese input x_i.

In the above setting, if the features $f(x_i, y)$ (e.g., the English-side n-gram features) are defined in a way such that they look at *only* the output y, we can train a discriminative language model. The downside of these approaches is that they rely on bilingual data for discriminative training. Note that all these models are globally normalized (as in the whole-sentence maximum entropy model of (5.2)), but avoid computing the very expensive $Z(*)$ that is required in (5.2).

5.3.4 Relation to Minimum Imputed Risk

Conceptually, the method proposed in this chapter is quite similar to the minimum imputed-risk training in Chapter 4 on page 56. In particular, both try to simulate the con-

fusion set of an observed English sentence $\tilde{y}$.

More concretely, the method here can be thought as an approximation to the intractable minimum imputed-risk objective of (4.8) on page 60. In particular, it is the same as the *rule-level composition* approximation discussed in Section 4.2.4.4 on page 61 except that our neighborhood is unweighted[6] (since the confusion rules are unweighted) and that we do not use a Chinese/English language model in the construction.[7] Due to these approximations, the method proposed in this chapter is much faster than the one in Section 4.2.4.4 (but not faster than the method used in the experiments there, which uses the k-best approximation described in Section 4.2.4.1). On the other hand, with these approximations, our method may do a less good job of approximating the actual confusion sets.[8] To address this limitation, we try to get closer back to the actual confusion sets by pruning back the confusion grammar to just the confusions that show up in the pruned hypergraphs generated from the Chinese test set (although see footnote 3).

5.3.5 Relation to Paraphrasing Models

Our method is also related to those of training paraphrasing models (Quirk, Brockett, and Dolan, 2004; Bannard and Callison-Burch, 2005; Callison-Burch, Koehn, and Osborne, 2006a; Madnani, Ayan, Resnik, and Dorr, 2007). Specifically, the form of our *confusion grammar* is similar to that of the *paraphrase model* they use, and the way of extracting the grammar/model is also similar as both are derived by using a language (e.g., Chinese in our case) as a *pivot*. However, while a "translation" rule in a paraphrase model is expected to contain a pair of phrases (i.e., source- and target-side phrases) that are paraphrases (i.e., good alternates) of each other, a confusion rule in our confusion grammar may contain a pair of phrases that are typical bad alternates of each other proposed by an MT system.

The motivations and goals are also different. For example, the goal of Bannard and Callison-Burch (2005) is to extract paraphrases with the help of parallel corpora. Callison-Burch et al. (2006a) aim to improve MT quality by adding paraphrases in the translation table, while Madnani et al. (2007) aim to improve the minimum error rate training by adding the automatically generated paraphrases into the English reference sets. In contrast, our motivation is to train a better *discriminative* language model (by using the confusion grammar to decide what alternates the model should learn to discriminative), which will then be used to help MT.

[6] This is also true in contrastive estimation (Smith and Eisner, 2005).

[7] Note that the experiments in Chapter 4 also uses an approxiamtion, that is., the k-best approximation described in Section 4.2.4.1.

[8] It is worth emphasizing our unweighted neighborhood is still better than just Σ^*, which is what a generative LM would use.

Data	Purpose	# of sentences	
		Chinese	English
Set0	training baseline translation/language models	40k	40k
Set1	discriminative training (for relative weights among models)	1006	1006×16
	contrastive language model training**		
Set2	testing	506	506×16

Table 5.1: **three data sets for experiments.** In the column of "Purpose", ** indicates *only* the English data is needed for training. Note that for each Chinese sentence in *Set1* and *Set2*, there are 16 English references.

5.4 Experimental Results

We report results on a Chinese-to-English translation task using **Joshua** (Li et al., 2009a).

5.4.1 Data Sets

We use three data sets (see Table 5.1): one to build a baseline translation/language model, one to perform discriminative training (to find relative weights among models) and perform contrastive language model estimation, and one for testing.

5.4.2 Baseline MT System

Our translation model is built on a bilingual corpus (i.e., *Set0* of Table 5.1) for IWSLT 2005 Chinese to English translation task (Eck and Hori, 2005) and it consists of 40k pairs of transcribed utterances in the travel domain.[9] We use a *5-gram* language model with modified Kneser-Ney smoothing (Chen and Goodman, 1998), trained on the English side of *Set0* of Table 5.1.

The baseline MT system includes ten features that are standard in Hiero (Chiang, 2007). In particular, these include one baseline *language model* feature, three baseline *translation model* features, one *word penalty* feature, and five *arity features* (three to count how many rules with an arity of zero/one/two are used in a derivation, and two to count how many times the unary and binary glue rules are used in a derivation). The relative weights are tuned using minimum risk training on the *bilingual* data *Set1* of Table 5.1.

[9]This task is relatively small, compared with a more realistic task (e.g., the NIST one). We use such a task for computational efficiency.

5.4.3 Training Contrastive Language Models

We extract a confusion grammar (CG) from the bilingual grammar (that is specific to test data *Set1* and *Set2* of Table 5.1). The number of rules in the bilingual and confusion grammar are about 167k and 1583k, respectively. The CG will be the "translation" model for an English-to-English "translation" system. It defines the hypothesis space (before pruning). Note that the rules in CG is unweighted.

We train several contrastive models (with different features described below) on the *English* data of *Set1*, by using the iterative training procedure described at the end of Section 5.2.3. Specifically, at each iteration, we use the current contrastive model to generate a pruned hypergraph (i.e., a contrastive set) for each English sentence of *Set1*. These hypergraphs will be used to train a new contrastive model, which will be used for decoding in the next iteration. Note that each English sentence in *Set1* has 15 paraphrase sentences. We generate a separate hypergraph for each English sentence (say $\tilde{y}$), but for each such hypergraph we use both $\tilde{y}$ and its 15 paraphrase sentences as the corresponding references during training. [10]

We consider two classes of features:

- **Regular Language Model Features**: We consider two regular language model features: one *baseline 5-gram language model* feature ("BLM") and one *word penalty* feature ("WP").

- **Target-rule Bigram Features ("RuleBigram")**: For each confusion rule, we extract bigram features over the target-side symbols (including non-terminals and terminals). For example, if a confusion rule's target side is "*on the* X_1 *issue of* X_2" where X_1 and X_2 are non-terminals (with a position index), we can extract bigram features including: "*on the*", "*the* X", "X *issue*", "*issue of*", and "*of* X". Note that the index under the nonterminal of the rule has been removed in the features. We consider only those terminal symbols (i.e., regular English words) that occur frequently in the English text. Moreover, for the terminal symbols, we will use their dominant POS tags (instead of the symbol itself). We use 525 such bigram features. [11]

5.4.4 Results on Monolingual Simulation

We are first interested in seeing how our contrastive language model (CLM) performs as a language model itself. One usually uses the perplexity of the LM on some unseen English data to measure the goodness of an LM. We are mainly interested in how the model performs in picking a good English sentence *within its neighborhood* (defined by the confusion grammar). The test is performed as follows. For each test English sentence

[10] In practice, one may not have multiple references. We take the unrealistic advantage of our data set (which has 16 references) for stability of the minimum risk training.

[11] Note that the number of features here is smaller than that of Section 4.4.2 on page 65 since the features here remove the indices of the non-terminals in the rules.

System ID	Features			BLEU
	BLM	WP	RuleBigram	
1	✓			12.8
2	✓	✓		14.2
3	✓	✓	✓	25.3

Table 5.2: **BLEU scores on English test set.** Each system uses a different set of features (see Section 5.4.3) in the contrastive estimation. In the table, we use"BLM" to denote the regular n-gram language model, "WP" to denote the word penalty feature, and "RuleBigram" to denote the target-rule bigram features.

$\tilde{y}$ of *Set2*, the confusion grammar defines a full contrastive set $\mathcal{N}(\tilde{y})$ (i.e., an un-pruned hypergraph). We use a contrastive model to pick the best y from $\mathcal{N}(\tilde{y})$, and then compute its BLEU score by using $\tilde{y}$ and its 15 paraphrase sentences as references.

Table 5.2 shows the results under a regular n-gram LM and different CLMs (which vary in the features used during contrastive training). (Note that a CLM includes the baseline n-gram LM (i.e., BLM) as a feature.) Clearly, the CLMs perform better than the regular n-gram model (i.e., system-1), in terms of picking one-best from a neighborhood. These show that our contrastive language model performs well as a language model alone.

Note that the BLEU scores in Table 5.2 are relatively small (e.g., compared with those scores that will be shown in Table 5.3). This is mainly because the CLM model, which is the only model used to rank the hypotheses in the neighborhood (i.e., a hypergraph), looks at only the target-side of the confusion rules. In other words, the "channel" model that will look at both the source- and target-side of a confusion rule is missing in the English-to-English "translation" system.

5.4.5 Results on MT Test Data

We are also interested in seeing how the contrastive language model (CLM) performs during actual MT decoding. To incorporate the CLM into MT decoding, we add the model as an additional feature in the baseline system, and then tune the relative weights (now, we have *eleven* weights, instead of ten as in the baseline MT system) on the *bilingual* data *Set1* of Table 5.1.

Table 5.3 shows the results for several systems. Note that the MT system already has its own "BLM" and "WP" features, as discussed in Section 5.4.2. Therefore, when integrating a CLM model into MT decoding, we only use its "RuleBigram" features. However, these "RuleBigram"features might get trained along with other features during contrastive estimation, as we have shown in Table 5.2. Table 5.3 shows that the CLM helps to improve the baseline MT system.

System ID	Features			BLEU
	BLM	WP	**RuleBigram**	
baseline	not applicable			48.1
3	✓	✓	✓	49.5

Table 5.3: **BLEU scores on MT test set.** The baseline MT system has ten models/features, and the other system has eleven models/features, where the additional model is the contrastive language model (CLM). Note that, for the CLM, only the "RuleBigram" features will be used in the MT decoding, but these features get trained along with other features (e.g., the BLM) during the contrastive estimation.

5.4.6 Goodness of the Simulated Neighborhood

In supervised training, for a given Chinese input sentence, the forward system will generate many English sentences, among which the training will learn to discriminate. This set of English sentences can be thought as a neighborhood of the English reference translation for the given Chinese input. In the unsupervised case, the Chinese input is missing, and so the contrastive LM applies the confusion grammar to "translate" each observed English sentence to many alternative English sentences. This set can be thought as a *simulated* neighborhood of the original English sentence. For the CLM to work well, we expect that the simulated neighborhood will have some overlap with the true neighborhood that is generated from the true Chinese input.

To measure the goodness of the simulated neighborhood, we obtain the sets of n-gram types in the two neighborhoods, and then compute the ratio between the number of n-gram types in the intersection and that in the union. Figure 5.2 shows the results when comparing different sizes of k-best English strings in the neighborhoods. In general, the ratio decreases when n is larger. Also, the ratio does not change much along with the change of k.

Table 5.4 presents the simulated neighborhood's precisions and recalls of the n-grams in the true neighborhood. The n-grams are collected from k-best strings where $k = 100$, which corresponds to the first data point in Figure 5.2. We would conclude that the simulated neighborhood does a reasonably well job in simulating the true neighborhood.

5.4.7 Some Translation Examples

Table 5.5 presents several examples of translations which the system incorporating a contrastive LM (i.e., the system3 in Table 5.3) performs better than the baseline system in Table 5.3.

n-gram	Precision	Recall
unigram	36.5%	48.2%
bigram	10.1%	12.8%
trigram	3.7%	4.6%
4-gram	2.0%	2.4%

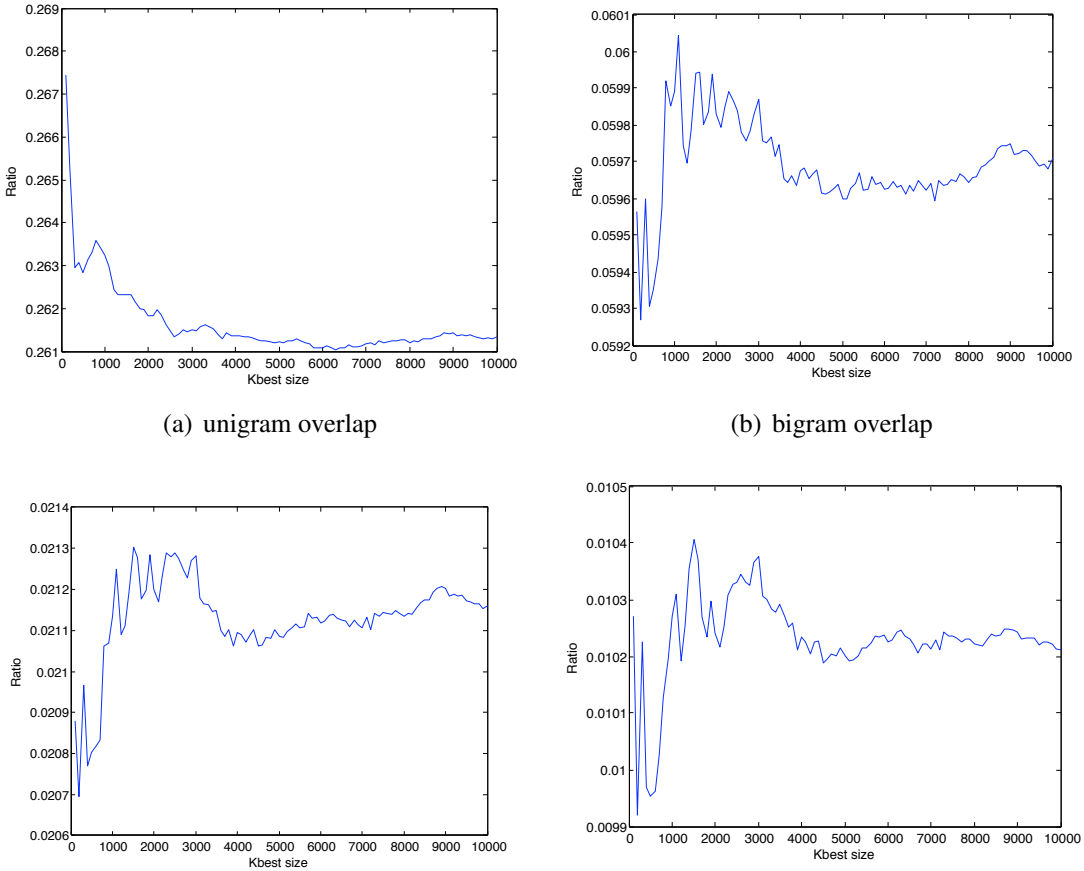
Table 5.4: Precisions and recalls of simulated neighborhood's n-grams. The n-grams are collected from k-best strings where $k = 100$, which corresponds to the first data point in Figure 5.2.

System	Output
Chinese	预约 柜台 在 哪 ?
Reference	where is the reservation counter ?
Baseline	where did counter ?
WithCLM	where is the reservation counter ?
Chinese	为什么 ? 我 已经 预订 了 。
Reference	why ? i made a reservation .
Baseline	why ? i 've reserved .
WithCLM	i have a reservation . why ?
Chinese	到 成田 机场 还 要 多 长 时间 ?
Reference	how much longer will it take to get to narita ?
Baseline	how long does it take to narita airport ?
WithCLM	how long will it take to narita airport ?
Chinese	恐怕 这 趟 航班 已经 订满 了 。
Reference	i 'm afraid this flight is booked solid .
Baseline	afraid fully booked on this flight .
WithCLM	afraid this flight is fully booked .
Chinese	你 认识 她 多久 了 ?
Reference	how long have you known her ?
Baseline	you know how her ?
WithCLM	how long do you know her ?

Table 5.5: Examples of translation outputs which the system incorporating a contrastive LM (i.e., the system3 in Table 5.3) performs better than the baseline system in Table 5.3.

5.4.8 Comparison to the Experiments in Section 4.4

The setup and the use of data sets are quite different between this section and Section 4.4 in Chapter 4. In particular, in Chapter 4, we split the 1006 sentences into two equal sets,



(a) unigram overlap

(b) bigram overlap

(c) trigram overlap

(d) 4-gram overlap

Figure 5.2: **The goodness of the simulated neighborhood by using confusion grammar.** We compare the two neighborhoods, one is generated from the Chinese input by the supervised baseline in Table 5.3, and the other is the simulated neighborhood generated from the English by using the CLM with a system ID of 3 in Table 5.2. To measure the goodness of the simulated neighborhood, we obtain the sets of n-gram types in the two neighborhoods, and then compute the ratio between the number of n-gram types in the intersection and that in the union. Finally, we obtain the average ratio among different sentences in the test set (i.e., *set3*).

one to train the reverse model, and the other to train the forward model (in a supervised, unsupervised, or semi-superivsed manner). In comparison, in this chapter we do not require a reverse model, so all 1006 sentences are used for training the forward system. Below, we give a rough comparison.

Best BLEU Score

The best BLEU score in Section 4.4 is 49.7 as shown in Table 4.2 on page 66, while the best BLEU score is 49.5 as shown in Table 5.3.

Goodness of the Simulated Neighborhoods

One can compare the goodness of the two different simulated neighborhoods by comparing between Figure 4.1 on page 70 and Figure 5.2 on page 86, and between Table 4.6 on page 69 and Table 5.4 on page 85. As we can see, these two simulated neighborhoods have similar quality.

Running Time

In terms of running time of the two kinds of experiments, they are similar, although the pipleline of the imputed-risk training will be a little bit more involved as we also need to train the reverse model.

5.5 Summary

We apply a similar idea as contrastive estimation to train a globally normalized log-linear language model for MT. Our method relies on a confusion grammar, which is an English-to-English SCFG and captures the confusion that an MT system may have when choosing different translations for a given input. We derive such a grammar from a bilingual grammar. For each English sentence, we then use the confusion grammar to generate a contrastive set, from which we train a contrastive language model (CLM). Our experiments show that the CLM is able to help a regular n-gram LM to pick a better one-best within a neighborhood of alternative sentences that are generated for an English sentence in a test set. The CLM also improves the performance of an MT system.

Chapter 6

Variational Decoding

As discussed in Section 2.5, due to spurious ambiguity, the maximum a posterior (MAP) decoding of (2.7) on page 28 is intractable. The Viterbi and crunching methods described there approximate the intractable decoding by ignoring most of the derivations. In this chapter, we will present a novel variational approximation, which considers all the derivations but still allows tractable decoding. Much of this chapter is based on Li et al. (2009b).

Our variational decoding works as follows. Given an input string, the original system produces a probability distribution p over possible output strings and their derivations. Our method constructs a second distribution $q \in \mathcal{Q}$ that approximates p as well as possible, and then finds the best string according to q. The last step is tractable because each $q \in \mathcal{Q}$ is defined (unlike p) without reference to the hidden derivations. Notice that q here does not approximate the entire translation process, but only the distribution over output strings *for a particular input*. This is why it can be a fairly good approximation even without looking at the hidden derivations.

In practice, we approximate with several different variational families $\mathcal{Q}$, corresponding to n-gram (Markov) models of different orders. We geometrically interpolate the resulting approximations q with one another (and with the Viterbi approximation of the original distribution p), justifying this interpolation as similar to the minimum-risk decoding for BLEU proposed by Tromble et al. (2008). Experiments show that our approach improves the state of the art.

The methods presented in this chapter should be applicable to collapsing spurious ambiguity for other tasks as well. Such tasks include data-oriented parsing (DOP), applications of Hidden Markov Models (HMMs) and mixture models, and other models with latent variables. Indeed, our methods were inspired by past work on variational decoding for DOP (Goodman, 1996) and for latent-variable parsing (Matsuzaki, Miyao, and Tsujii, 2005).

6.1 Variational Decoding for MT

Variational methods generally work as follows. When exact inference under a complex model p is intractable, one can approximate the posterior $p(y \mid x)$ by a tractable model $q(y)$, where $q \in \mathcal{Q}$ is chosen to minimize some information loss such as the KL divergence $\mathrm{KL}(p \,\|\, q)$. The simpler model q can then act as a surrogate for p during inference. Below we will discuss how to apply this general method to solve the intractable MAP decoding problem of (2.7) on page 28. In the discussion, we will use the notations defined in Table 2.1 on page 18.

For each input sentence x, we assume that a baseline MT system generates a hypergraph $\mathrm{HG}(x)$ that defines a probability distribution $p(d \mid x)$ over the derivations $d \in \mathrm{D}(x)$.[1] For any single $y \in \mathrm{T}(x)$, it would be tractable using $\mathrm{HG}(x)$ to compute $p(y \mid x) = \sum_{d \in \mathrm{D}(x,y)} p(d \mid x)$. However, as mentioned, it is intractable to find $\mathrm{argmax}_y\, p(y \mid x)$ as required by the MAP decoding (2.7) on page 28, so we seek an *approximate* distribution $q(y) \approx p(y \mid x)$.[2]

For a given x, we seek a distribution $q \in \mathcal{Q}$ that minimizes the KL divergence from p to q (both regarded as distributions over y):[3]

$$q^* = \operatorname*{argmin}_{q \in \mathcal{Q}} \mathrm{KL}(p \,\|\, q) \tag{6.1}$$

$$= \operatorname*{argmin}_{q \in \mathcal{Q}} \sum_{y \in \mathrm{T}(x)} (p \log p - p \log q) \tag{6.2}$$

$$= \operatorname*{argmax}_{q \in \mathcal{Q}} \sum_{y \in \mathrm{T}(x)} p \log q \tag{6.3}$$

So far, in order to approximate the intractable optimization problem (2.7), we have defined another optimization problem (6.3). If computing $p(y \mid x)$ during decoding is computationally intractable, one might wonder if the optimization problem (6.3) is any simpler. We will show this is the case. The trick is to **parameterize** q as a factorized distribution such that the **estimation** of q^* and **decoding** using q^* are both tractable through efficient dynamic programs. In the next three subsections, we will discuss the *parameterization*, *estimation*, and *decoding*, respectively.

6.1.1 Parameterization of q

In (6.3), $\mathcal{Q}$ is a family of distributions. If we select a large family $\mathcal{Q}$, we can allow more complex distributions, so that q^* will better approximate p. If we select a smaller family $\mathcal{Q}$,

[1] The baseline system may return a pruned hypergraph, which has the effect of pruning $\mathrm{D}(x)$ and $\mathrm{T}(x)$ as well.

[2] Following the convention in describing variational inference, we write $q(y)$ instead of $q(y \mid x)$, even though $q(y)$ always depends on x implicitly.

[3] To avoid clutter, we denote $p(y \mid x)$ by p, and $q(y)$ by q. We drop $p \log p$ from (6.2) because it is constant with respect to q. We then flip the sign and change argmin to argmax.

we can guarantee that q^* will have a simple form with many conditional independencies, so that $q^*(y)$ and $y^* = \text{argmax}_y\, q^*(y)$ are easier to compute.

Since each $q(y)$ is a distribution over output strings, a natural choice for $\mathcal{Q}$ is the family of n-gram models. To obtain a small KL divergence (6.1), we should make n as large as possible. In fact, $q^* \to p$ as $n \to \infty$. Of course, this last point also means that our computation becomes intractable as $n \to \infty$.[4] However, if $p(y \mid x)$ is defined by a hypergraph $\text{HG}(x)$ whose structure explicitly incorporates an m-gram language model, both training and decoding will be efficient when $m \geq n$. We will give algorithms for this case that are linear in the size of $\text{HG}(x)$.[5]

Formally, each n-gram model $q_n \in \mathcal{Q}$ takes the form

$$q_n(y) = \prod_{w \in W_n} q(r(w) \mid h(w))^{c_w(y)} \tag{6.4}$$

where W_n is a set of n-gram types. Each $w \in W$ is an n-gram, which occurs $c_w(y)$ times in the string y, and w may be divided into an $(n-1)$-gram prefix $h(w)$ (the *history*) and a 1-gram suffix $r(w)$ (the *rightmost* or *current* word).

The parameters that specify a particular $q_n \in \mathcal{Q}$ are the (normalized) conditional probability distributions $q(r(w) \mid h(w))$. We will now see how to estimate these parameters to approximate $p(\cdot \mid x)$ for a given x at test time.

---

[4] Blunsom et al. (2008) effectively do take $n = \infty$, by maintaining the whole translation string in the dynamic programming state. They alleviate the computation cost somehow by using aggressive beam pruning, which might be sensible for their relatively small task (e.g., input sentences of < 10 words) like IWSLT. But, we are interested in improving the performance for a large-scale system, and thus their method is not a viable solution. Moreover, we observe in our experiments that using a larger n does not improve much over $n = 2$.

[5] One might ask how our method interacts with backed-off language models. The issue is that the most compact finite-state representations of these (Allauzen et al., 2003), which exploit backoff structure, are not purely m-gram for any m. They yield more compact hypergraphs (Li and Khudanpur, 2008a), but unfortunately those hypergraphs might not be treatable by Fig. 6.2—since where they back off to less than an n-gram, e is not informative enough for line 8 to find w.

We sketch a method that works for *any* language model given by a weighted FSA, L. The variational family $\mathcal{Q}$ can be specified by any deterministic weighted FSA, Q, with weights parameterized by ϕ. One seeks ϕ to minimize (6.1).

Intersect $\text{HG}(x)$ with an "unweighted" version of Q in which all arcs have weight 1, so that Q does not prefer any string to another. By lifting weights into an expectation semiring (Eisner, 2002), it is then possible to obtain expected transition counts in Q (where the expectation is taken under p), or other sufficient statistics needed to estimate ϕ.

This takes only time $O(|\text{HG}(x)|)$ when L is a left-to-right refinement of Q (meaning that any two prefix strings that reach the same state in L also reach the same state in Q), for then intersecting L or $\text{HG}(x)$ with Q does not split any states. That is the case when L and Q are respectively pure m-gram and n-gram models with $m \geq n$, as assumed in (6.5) and Figure 6.2. It is also the case when Q is a pure n-gram model and L is constructed not to back off beyond n-grams; or when the variational family $\mathcal{Q}$ is defined by deliberately taking the FSA Q to have the same topology as L.

6.1.2 Estimation of q^*

Note that the objective function (6.1)–(6.3) asks us to approximate p as closely as possible, without any further smoothing. (It is assumed that p is already smoothed appropriately, having been constructed from channel and language models that were estimated with smoothing from finite training data.)

In fact, if p were the empirical distribution over strings in a training corpus, then q^* of (6.3) is just the maximum-likelihood n-gram model—whose parameters, trivially, are just unsmoothed ratios of the n-gram and $(n-1)$-gram counts in the training corpus. That is, $q^*(r(w) \mid h(w)) = \frac{c(w)}{c(h(w))}$.

Our actual job is exactly the same, except that p is specified not by a corpus but by the hypergraph $\mathrm{HG}(x)$. The only change is that the n-gram counts $\bar{c}(w)$ are no longer integers from a corpus, but are expected counts under p:[6]

$$\begin{aligned} q^*(r(w) \mid h(w)) &= \frac{\bar{c}(w)}{\bar{c}(h(w))} \\ &= \frac{\sum_y c_w(y) p(y \mid x)}{\sum_y c_{h(w)}(y) p(y \mid x)} \\ &= \frac{\sum_d c_w(\mathrm{Y}(d)) p(d \mid x)}{\sum_d c_{h(w)}(\mathrm{Y}(d)) p(d \mid x)} \end{aligned} \tag{6.5}$$

Now, the question is how to efficiently compute (6.5) from the hypergraph $\mathrm{HG}(x)$. To develop the intuition, we first present a brute-force algorithm in Figure 6.1. The algorithm is brute-force since it first needs to unpack the hypergraph and enumerate each possible derivation in the hypergraph (see line 1), which is computationally intractable. The algorithm then enumerates each n-gram and $(n-1)$-gram in y and accumulates its soft count into the expected count, and finally obtains the parameters of q^* by taking count ratios via (6.5).

Figure 6.2 shows an efficient version that exploits the packed-forest structure of $\mathrm{HG}(x)$ in computing the expected counts. Specifically, it first runs the inside-outside procedure, which annotates each node (say v) with both an *inside weight* $\beta(v)$ and an *outside weight* $\alpha(v)$. The inside-outside also finds $\mathrm{Z}(x)$, the total weight of all derivations. With these weights, the algorithm then explores the hypergraph once more to collect the expected counts. For each hyperedge (say e), it first gets the *posterior weight* c_e (see lines 4-6). Then, for each n-gram type (say w), it increments the expected count by $c_w(e) \cdot c_e$, where $c_w(e)$ is the number of copies of n-gram w that are added by hyperedge e, i.e., that appear in the yield of e but not in the yields of any of its antecedents $u \in T(e)$.

While there may be exponentially many derivations, the hypergraph data structure represents them in polynomial space by allowing multiple derivations to share subderivations.

[6] One can prove (6.5) via Lagrange multipliers, with $q^*(\cdot \mid h)$ constrained to be a normalized distribution for each h.

Brute-Force-MLE$(\mathrm{HG}(x))$

```
for d in HG(x)    ▷ each derivation
  for w in Y(d)    ▷ each n-gram type in the yilt of d
      ▷ accumulate soft count
      c̄(w)  +=  c_w(Y(d)) · p(d | x)
      c̄(h(w))  +=  c_w(Y(d)) · p(d | x)
q* ← MLE using formula (6.5)
return q*
```

Figure 6.1: Brute-force estimation of q^*.

Dynamic-Programming-MLE$(\mathrm{HG}(x))$

```
 1  run inside-outside on the hypergraph HG(x)
 2  for v in HG(x)    ▷ each node
 3    for e ∈ I(v)    ▷ each incoming hyperedge
 4      c_e ← p_e · α(v)/Z(x)
 5      for u ∈ T(e)    ▷ each antecedent node
 6          c_e ← c_e · β(u)
 7      ▷ accumulate soft count
 8      for w in e    ▷ each n-gram type
 9          c̄(w)  +=  c_w(e) · c_e
10          c̄(h(w))  +=  c_w(e) · c_e
11  q* ← MLE using formula (6.5)
12  return q*
```

Figure 6.2: Dynamic programming estimation of q^*. $I(v)$ represents the set of incoming hyperedges of node v; p_e represents the weight of the hyperedge e itself; $T(e)$ represents the set of antecedent nodes of hyperedge e. Please refer to the text for the meanings of other notations.

The algorithm of Figure 6.2 may be run over this packed forest in time $\mathrm{O}(|\mathrm{HG}(x)|)$ where $|\mathrm{HG}(x)|$ is the hypergraph's size (number of *hyperedges*).

Note that the lines 1–10 of Figure 6.2 is a specific version of the general inside-outside algorithm of Figure 3.3 on page 42. Indeed, the expected n-gram counts can be computed by using a first-order expectation semiring in the algorithm of Figure 3.3, where the weight for a hyperedge e is $\langle p_e, p_e c_w(e) \rangle$.

6.1.3 Decoding with q^*

When translating x at runtime, the q_n^* constructed from $\text{HG}(x)$ will be used as a surrogate for p during decoding. We want its most probable string:

$$y^* = \operatorname*{argmax}_y q_n^*(y) \tag{6.6}$$

Since q_n^* is an n-gram model, finding y^* is equivalent to a shortest-path problem in a certain graph whose edges correspond to n-grams (weighted with negative log-probabilities) and whose vertices correspond to $(n-1)$-grams.

However, because q_n^* only approximates p, y^* of (6.6) may be locally appropriate but globally inadequate as a translation of x. Observe, e.g., that an n-gram model $q_n^*(y)$ will tend to favor short strings y, regardless of the length of x. Suppose $x =$ *le chat chasse la souris* ("the cat chases the mouse") and q^* is a bigram approximation to $p(y \mid x)$. Presumably $q^*(\text{the} \mid \text{START})$, $q^*(\text{mouse} \mid \text{the})$, and $q^*(\text{END} \mid \text{mouse})$ are all large in $\text{HG}(x)$. So the most probable string y^* under q^* may be simply "the mouse," which is short and has a high probability but fails to cover x.

Therefore, a better way of using q^* is to restrict the search space to the original hypergraph, i.e.:

$$y^* = \operatorname*{argmax}_{y \in \text{T}(x)} q_n^*(y) \tag{6.7}$$

This ensures that y^* is a valid string in the original hypergraph $\text{HG}(x)$, which will tend to rule out inadequate translations like "the mouse."

If our sole objective is to get a good approximation to $p(y \mid x)$, we should just use a *single* n-gram model q_n^* whose order n is as large as possible, given computational constraints. This may be regarded as favoring n-grams that are likely to appear in the reference translation (because they are likely in the derivation forest). However, in order to score well on the BLEU metric for MT evaluation, which gives partial credit, we would also like to favor lower-order n-grams that are likely to appear in the reference, even if this means picking some less-likely high-order n-grams. For this reason, it is useful to interpolate different orders of variational models,

$$y^* = \operatorname*{argmax}_{y \in \text{T}(x)} \sum_n \theta_n \cdot \log q_n^*(y) \tag{6.8}$$

where n may include the value of zero, in which case $\log q_0^*(y) \stackrel{\text{def}}{=} |y|$, corresponding to a conventional word penalty feature. In the *geometric* interpolation above, the weight θ_n controls the relative veto power of the n-gram approximation and can be tuned by any discriminative method presented in Section 2.6 on page 30.

Lastly, note that Viterbi and variational approximation are different ways to approximate the exact probability $p(y \mid x)$, and each of them has pros and cons. Specifically, Viterbi approximation (see Section 2.5.1 on page 29) uses the correct probability of one

complete derivation, but ignores most of the derivations in the hypergraph. In comparison, the variational approximation considers all the derivations in the hypergraph, but uses only aggregate statistics of fragments of derivations. Therefore, it is desirable to interpolate further with the Viterbi approximation when choosing the final translation output:[7]

$$y^* \quad = \quad \operatorname*{argmax}_{y \in \mathrm{T}(x)} \theta_v \cdot \log p_{\mathrm{Viterbi}}(y \mid x) + \sum_n \theta_n \cdot \log q_n^*(y) \tag{6.9}$$

where the first term corresponds to the Viterbi decoding of (2.8) on page 29 and the second term corresponds to the interpolated variational decoding of (6.8).[8] Assuming $\theta_v > 0$, the first term penalizes translations with no good derivation in the hypergraph.[9]

For $n \leq m$, any of these decoders (6.7)–(6.9) may be implemented efficiently by using the n-gram variational approximations q^* to rescore $\mathrm{HG}(x)$—preserving its hypergraph topology, but modifying the hyperedge weights.[10] While the original weights gave derivation d a score of $\log p(d \mid x)$, the weights as modified for (6.9) will give d a score of $\theta_v \cdot \log p(d \mid x) + \sum_n \theta_n \cdot \log q_n^*(\mathrm{Y}(d))$. We then find the best-scoring derivation and output its target yield; that is, we find $\operatorname{argmax}_{y \in \mathrm{T}(x)}$ via $\mathrm{Y}(\operatorname{argmax}_{d \in \mathrm{D}(x)})$.

6.2 Variational vs. Minimum Bayes Risk Decoding

In a high-level, variational and minimum Bayes risk (MBR) decoding aim to solve different problems as shown in Figure 6.3: one is to approximate the intractable MAP decoding (due to spurious ambiguity), and the other is to find a consensus translation.

Now, we formally derive the connection between our variational decoding and the MBR decoding of Tromble et al. (2008). Recall that the MBR decision rule is,

$$y^* = \operatorname*{argmin}_y \mathrm{R}(y) = \operatorname*{argmin}_y \sum_{y'} \mathrm{L}(y, y') p(y' \mid x) \tag{6.10}$$

[7] It would also be possible to interpolate with the k-best approximation (see Section 2.5.1), with some complications.

[8] Zens and Ney (2006) use a similar decision rule as here and they also use posterior n-gram probabilities as feature functions, but their model estimation and decoding are over a k-best, which is trivial in terms of computation.

[9] Already at (6.7), we explicitly *ruled out* translations y having *no derivation at all* in the hypergraph. However, suppose the hypergraph were very large (thanks to a large or smoothed translation model and weak pruning). Then (6.7)'s heuristic would fail to eliminate bad translations ("the mouse"), since nearly *every* string $y \in \Sigma^*$ would be derived as a translation with at least a tiny probability. The "soft" version (6.9) solves this problem, since unlike the "hard" (6.7), it penalizes translations that appear only weakly in the hypergraph. As an extreme case, translations not in the hypergraph at all are infinitely penalized ($\log p_{\mathrm{Viterbi}}(y) = \log 0 = -\infty$), making it natural for the decoder not to consider them, i.e., to do only $\operatorname{argmax}_{y \in \mathrm{T}(x)}$ rather than $\operatorname{argmax}_{y \in \Sigma^*}$.

[10] One might also want to use the q_n^* or smoothed versions of them to rescore additional hypotheses, e.g., hypotheses proposed by other systems or by system combination.

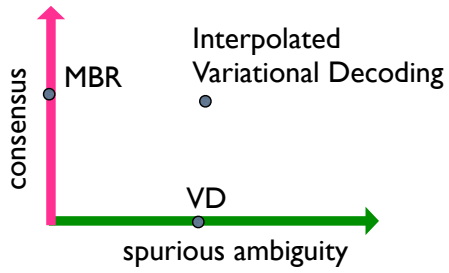


Figure 6.3: MBR decoding versus variational decoding. While the variational decoding (VD) is mainly to approximate the intractable MAP decoding problem due to spurious ambiguity, the main goal of minimum Bayes risk (MBR) decoding is to find a consensus translation. In comparison, the VD decoding with interpolations of different order of n-gram approximations addresses both the spurious ambiguity and consensus problems.

where $\mathrm{L}(y, y')$ represents the loss of y if the true answer is y', and the **risk** of y is its expected loss. As mentioned before, the MBR decoding is intractable if we use the negated BLEU (Papineni et al., 2001) as a loss function. Therefore, Tromble et al. (2008) use the following loss function, of which a linear approximation to BLEU a special case,

$$\mathrm{L}(y, y') = -(\theta_0|y| + \sum_{w \in N} \theta_w c_w(y) \delta_w(y')) \tag{6.11}$$

where w is an n-gram type, N is a set of n-gram types with $n \in [1, 4]$, $c_w(y)$ is the number of occurrence of the n-gram w in y, and $\delta_w(y')$ is an indicator function to check if y' contains at least one occurrence of w. With the above loss function, Tromble et al. (2008) derive the MBR rule[11]

$$y^* = \operatorname*{argmax}_y (\theta_0|y| + \sum_{w \in N} \theta_w c_w(y) g(w \mid x)) \tag{6.12}$$

where $g(w \mid x)$ is a specialized "posterior" probability of the n-gram w, and is defined as

$$g(w \mid x) = \sum_{y'} \delta_w(y') p(y' \mid x) \tag{6.13}$$

Now, let us divide N, which contains n-gram types of different n, into several subsets W_n, each of which contains only the n-grams with a given length n. We can now rewrite (6.12) as follows,

$$y^* = \operatorname*{argmax}_y \sum_n \theta_n \cdot g_n(y \mid x) \tag{6.14}$$

[11] Note that Tromble et al. (2008) only consider MBR for a lattice without hidden structures, though their method can be in principle applied in a hypergraph with spurious ambiguity.

Decoding	MBR	Variational (Interpolated)
Decision rule	$y^* = \operatorname{argmax}_y \sum_n \theta_n \cdot g_n(y)$	$y^* = \operatorname{argmax}_y \sum_n \theta_n \cdot \log q_n(y)$
n-gram model	$g_n(y) = \sum_{w \in W_n} g(w) c_w(y)$	$q_n(y) = \prod_{w \in W_n} q(r(w) \mid h(w))^{c_w(y)}$
n-gram probability	$g(w) = \sum_{y'} \delta_w(y') p(y')$	$q(r(w) \mid h(w)) = \frac{\sum_{y'} c_w(y') p(y')}{\sum_{y'} c_{h(w)}(y') p(y')}$

Table 6.1: MBR versus variational decoding. Note that all the probability of y (or y') should depend on a given input x, but we do not explicitly write that for convenience.

by assuming $\theta_w = \theta_{|w|}$ and,

$$g_n(y \mid x) = \begin{cases} |y| & \text{if } n = 0 \\ \sum_{w \in W_n} g(w \mid x) c_w(y) & \text{if } n > 0 \end{cases} \tag{6.15}$$

Clearly, their rule (6.14) has a quite similar form to our rule (6.8), and we can relate (6.13) to (6.5) and (6.15) to (6.4). This justifies the use of interpolation in Section 6.1.3. However, there are several important differences (see Table 6.1). First, the n-gram "posterior" of (6.13) is very expensive to compute. In fact, it requires an intersection between each n-gram in the lattice and the lattice itself, as is done by Tromble et al. (2008). In comparison, the optimal n-gram probabilities of (6.5) can be computed using the inside-outside algorithm, once and for all. Also, $g(w \mid x)$ of (6.13) is not normalized over the history of w, while $q^*(r(w) \mid h(w))$ of (6.5) is. Lastly, the definition of the n-gram model is different. While the model (6.4) is a proper probabilistic model, the function of (6.15) is simply an approximation of the average n-gram precisions of y.

A connection between variational decoding and minimum-risk decoding has been noted before (e.g., Matsuzaki et al. (2005)), but the derivation above makes the connection formal.

6.3 Experimental Results

We report results using an open source MT toolkit **Joshua** (Li et al., 2009a).

6.3.1 Experimental Setup

We work on a Chinese to English translation task. Our translation model was trained on about 1M parallel sentence pairs (about 28M words in each language), which are subsampled from corpora distributed by LDC for the NIST MT evaluation using a sampling

method based on the n-gram matches between training and test sets in the foreign side. We also used a *5-gram* language model with modified Kneser-Ney smoothing (Chen and Goodman, 1998), trained on a data set consisting of a 130M words in English Gigaword (LDC2007T07) and the English side of the parallel corpora. We use GIZA++ (Och and Ney, 2000), a suffix-array Hiero grammar extractor (Lopez, 2007), SRILM (Stolcke, 2002), and risk-based deterministic annealing (Smith and Eisner, 2006)[12] to obtain word alignments, translation models, language models, and the optimal weights for combining these models, respectively. We use standard beam-pruning and cube-pruning parameter settings, following Chiang (2007), when generating the hypergraphs.

The NIST MT'03 set is used to tune model weights (e.g. those of (6.9)) and the scaling factor γ of (2.17),[13] and MT'04 and MT'05 are blind test-sets. We will report results for lowercase BLEU-4, using the shortest reference translation in computing brevity penalty.

6.3.2 Main Results

Table 6.2 presents the BLEU scores under Viterbi of (2.8) on page 29, crunching of (2.11) on page 29, K-best MBR, and variational decoding. Both crunching and MBR show slight significant improvements over the Viterbi baseline; variational decoding gives a substantial improvement.

The difference between MBR and Crunching+MBR lies in how we approximate the distribution $p(y' \mid x)$ in (6.10).[14] For MBR, we take $p(y' \mid x)$ to be proportional to $p_{\text{Viterbi}}(y' \mid x)$ if y' is among the K best distinct *strings* on that measure, and 0 otherwise. For Crunching+MBR, we take $p(y' \mid x)$ to be proportional to $p_{\text{crunch}}(y' \mid x)$, which is based on the N best *derivations*.

6.3.3 Results of Different Variational Decoding

Table 6.3 presents the BLEU results under different ways in using the variational models, as discussed in Section 6.1.3. As shown in Table 6.3(a), decoding with a single variational n-gram model (**VM**) as per (6.7) improves the Viterbi baseline (except the case with a unigram VM), though often not statistically significant. Moreover, a bigram (i.e., "2gram") achieves the best BLEU scores among the four different orders of VMs.

The interpolation between a VM and a word penalty feature ("wp") improves over the unigram VM dramatically, but does not improve higher-order VMs (Table 6.3(b)). Adding the Viterbi feature ("vt") into the interpolation further improves the lower-order models

[12] We have also experimented with MERT (Och, 2003), and found that the deterministic annealing gave results that were more consistent across runs and often better.

[13] We found the BLEU scores are not very sensitive to γ, contrasting to the observations by Tromble et al. (2008).

[14] We also restrict $\mathrm{T}(x)$ to $\{y : p(y \mid x) > 0\}$, using the same approximation for $p(y \mid x)$ as we did for $p(y' \mid x)$.

Decoding scheme	MT'04	MT'05
Viterbi	35.4	32.6
MBR (K=1000)	35.8	32.7
Crunching (N=10000)	35.7	32.8
Crunching+MBR (N=10000)	35.8	32.7
Variational (1to4gram+wp+vt)	**36.6**	**33.5**

Table 6.2: BLEU scores for Viterbi, Crunching, MBR, and variational decoding. All the systems improve significantly over the Viterbi baseline (paired permutation test, $p < 0.05$). In each column, we **boldface** the best result as well as all results that are statistically indistinguishable from it. In MBR, K is the number of unique *strings*. For Crunching and Crunching+MBR, N represents the number of *derivations*. On average, each string has about 115 distinct derivations. The variational method "1to4gram+wp+vt" is our full interpolation (6.9) of four variational n-gram models ("1to4gram"), the Viterbi baseline ("vt"), and a word penalty feature ("wp").

(Table 6.3(c)), and all the improvements over the Viterbi baseline become statistically significant. At last, interpolation of several variational models does not yield much further improvement over the best previous model, but makes the results more stable (Table 6.3(d)).

6.3.4 KL Divergence of Approximate Models

While the BLEU scores reported show the practical utility of the variational models, it is also interesting to measure how well each individual variational model $q(y)$ approximates the distribution $p(y \mid x)$. Ideally, the quality of approximation should be measured by the KL divergence $\mathrm{KL}(p \,\|\, q) \stackrel{\text{def}}{=} \mathrm{H}(p, q) - \mathrm{H}(p)$, where the cross-entropy $\mathrm{H}(p, q) \stackrel{\text{def}}{=} -\sum_y p(y \mid x) \log q(y)$, and the entropy $\mathrm{H}(p) \stackrel{\text{def}}{=} -\sum_y p(y \mid x) \log p(y \mid x)$. Unfortunately $\mathrm{H}(p)$ (and hence $\mathrm{KL} = \mathrm{H}(p, q) - \mathrm{H}(p)$) is intractable to compute. But, since $\mathrm{H}(p)$ is the same for all q, we can simply use $\mathrm{H}(p, q)$ to compare different models q. Table 6.4 reports the cross-entropies $\mathrm{H}(p, q)$ for various models q.

We also report the *derivational* entropy $\mathrm{H}_d(p) \stackrel{\text{def}}{=} -\sum_d p(d \mid x) \log p(d \mid x)$.[15] From this, we obtain an estimate of $\mathrm{H}(p)$ by observing that the "gap" $\mathrm{H}_d(p) - \mathrm{H}(p)$ equals $\mathbf{E}_{p(y)}[\mathrm{H}(d \mid y)]$, which we estimate from our 10000-best list.

Table 6.4 confirms that higher-order variational models (drawn from a larger family Q) approximate p better. This is necessarily true, but it is interesting to see that most of the improvement is obtained just by moving from a unigram to a bigram model. Indeed, although Table 6.4 shows that better approximations can be obtained by using higher-order models, the best BLEU score in Tables 6.3(a) and 6.3(c) was obtained by the bigram model. After

[15] Both $\mathrm{H}(p, q)$ and $\mathrm{H}_d(p)$ involve an expectation over exponentially many derivations, but they can be computed in time only linear in the size of $\mathrm{HG}(x)$ using an *expectation semiring* (see Section 3.4) on page 50.

(a) decoding with a single variational model

Decoding scheme	MT'04	MT'05
Viterbi	35.4	32.6
1gram	25.9	24.5
2gram	36.1	33.4
3gram	36.0*	33.1
4gram	35.8*	32.9

(b) interpolation between a single variational model and a word penalty feature

1gram+wp	29.7	27.7
2gram+wp	35.5	32.6
3gram+wp	36.1*	33.1
4gram+wp	35.7*	32.8*

(c) interpolation of a single variational model, the Viterbi model, and a word penalty feature

1gram+wp+vt	35.6*	32.8*
2gram+wp+vt	**36.5***	**33.5***
3gram+wp+vt	35.8*	32.9*
4gram+wp+vt	35.6*	32.8*

(d) interpolation of several n-gram VMs, the Viterbi model, and a word penalty feature

1to2gram+wp+vt	36.6*	**33.6***
1to3gram+wp+vt	**36.6***	33.5*
1to4gram+wp+vt	**36.6***	33.5*

Table 6.3: BLEU scores under different variational decoders discussed in Section 6.1.3. A star * indicates a result that is significantly better than Viterbi decoding (paired permutation test, $p < 0.05$). We **boldface** the best system and all systems that are not significantly worse than it. The brevity penalty BP in BLEU is always 1, meaning that on average y^* is no shorter than the reference translation, except for the "1gram" systems in (a), which suffer from brevity penalties of 0.826 and 0.831.

all, p cannot perfectly predict the reference translation anyway, hence may not be worth approximating closely; but p may do a good job of predicting bigrams of the reference translation, and the BLEU score rewards us for those.

6.3.5 Some Translation Examples

Table 6.5 presents several examples of translations which the interpolated variational decoding (i.e., the system of the last row in Table 6.2) performs better than the Viterbi

Measure	$\overline{\mathrm{H}}(\boldsymbol{p}, \cdot)$				$\overline{\mathrm{H}_d}(\boldsymbol{p})$	$\overline{\mathrm{H}}(\boldsymbol{p})$
bits/word	q_1^*	q_2^*	q_3^*	q_4^*		$\approx$
MT'04	2.33	1.68	1.57	1.53	1.36	1.03
MT'05	2.31	1.69	1.58	1.54	1.37	1.04

Table 6.4: Cross-entropies $\overline{\mathrm{H}}(p, q)$ achieved by various approximations q. The notation $\overline{\mathrm{H}}$ denotes the sum of cross-entropies of all test sentences, divided by the total number of test words. A perfect approximation would achieve $\overline{\mathrm{H}}(p)$, which we *estimate* using the true $\overline{\mathrm{H}_d}(p)$ and a 10000-best list.

decoding (i.e., the system of the first row in Table 6.2).

6.4 Summary

We have successfully applied the general variational inference framework to a large-scale MT task, to approximate the intractable problem of MAP decoding in the presence of spurious ambiguity. We also showed that interpolating variational models with the Viterbi approximation can compensate for poor approximations, and that interpolating them with one another can reduce the Bayes risk and improve BLEU. Our empirical results improve the state of the art.

System	Output
Chinese	美 与 日韩 磋商 后 对 与 北韩 复谈 非常 乐观
Reference	u . s . optimistic on new talks with north korea after discussions with japan and south korea
Viterbi	the united states and south korea consultations to resume negotiations with north korea is very optimistic about
Variational	us after consultation with south korea is very optimistic about resuming talks with north korea
Chinese	卡苏里 高度 评价 中国 为 促进 南亚 地区 的 和平 与 发展 所 发挥 的 建设性 作用 。
Reference	kasuri spoke highly of china's constructive role in promoting peace and development in the south asia region .
Viterbi	kasuri spoke highly of china 's south asia to promote the constructive role played by the peace and development of the region .
Variational	kasuri spoke highly of china to promote peace and development in the south asian region played a constructive role .
Chinese	在 许多 公司 持续 整顿 以 确保 获利 之际 , 日本 的 失业率 仍 高于 百分之五 。
Reference	japan's jobless rate remains above five percent as many companies continue their restructuring efforts to secure profits .
Viterbi	in many companies continued to rectify the profits in order to ensure that japan 's unemployment rate , still higher than 5 percent .
Variational	in many companies continued rectification to ensure profits , japan 's unemployment rate is still higher than 5 percent .

Table 6.5: Examples of translation outputs which the interpolated variational decoding (i.e., the system of the last row in Table 6.2) performs better than the Viterbi decoding (i.e., the system of the first row in Table 6.2).

Chapter 7

Conclusion

In machine translation, a hypergraph can be used to compactly represent exponentially many possible translations (and their derivations as well). In this dissertation, we have presented novel training and decoding methods and new algorithms to carry out the needed computation on a hypergraph. More specifically, we have presented:

- **First- and second-order expectation semirings** (Chapter 3): These semirings can be used in a semiring parsing framework to compute many interesting expectations (e.g., risk and its gradient) over hypergraphs. They have enabled minimum-risk training over a hypergraph, which is used in both chapters of 4 and 5. The expectation semiring also enables computing the expected n-gram counts and (cross-)entropy that are used in Chapter 6.

- **Minimum imputed-risk training** (Chapter 4): This is an unsupervised method that can exploit monolingual English data to perform discriminative training. It uses a reverse translation model to impute the missing inputs, and then trains a discriminative forward model by minimizing the expected loss of the forward translations of the missing inputs.

- **Contrastive language model estimation** (Chapter 5): This is another unsupervised method, which can also exploit monolingual English data to perform discriminative training, but does not require a reverse system. It first extracts a confusion grammar, then generates a contrastive set for each English sentence using the confusion grammar, and finally trains a discriminative language model on the contrastive sets such that the model will prefer the original English sentences (over the sentences in the contrastive sets).

- **Variational decoding** (Chapter 6): It approximates the intractable maximum a posterior (MAP) decoding problem (the intractability is due to spurious ambiguity) using a variational principle. In particular, given an intractable distribution p, it estimates

a simpler distribution q (i.e., an n-gram model), which will serve as a surrogate of p during decoding.

There are several novel contributions presented in the dissertation. Specifically, the contributions of Chapter 3 are:

- extending the first-order expectation semiring, which was originally proposed by Eisner (2002) for an FSA, to a hypergraph,
- proposing a novel second-order expectation semiring,
- presenting an inside-outside speedup trick when the elements of the semirings are vectors or matrices,
- and presenting many examples/applications that use the semirings (see Table 3.4 on page 53).

The main contributions in Chapter 4 include:

- presenting the minimum imputed-risk objective,
- applying it to MT and presenting approximations that are required in the training,
- and performing supervised, unsupervised, and semi-supervised experiments.

The contributions in Chapter 5 include:

- presenting a pipeline to train a discriminative language model for MT,
- proposing the notion of confusion grammar, and presenting ways in extracting it,
- and performing experiments to verify the goodness of a contrastive language model.

Finally, Chapter 6 includes contributions such as,

- presenting a variation inference procedure to approximate the intractable MAP decoding,
- formally deriving the connection between variational and MBR decoding,
- and performing experiments to show that variational decoding improves state of the art MT performance.

7.1 Future Work

We have already exploited the first- and second-order expectation semirings to enable several interesting applications. For example, the core computation of minimum risk training (used in both chapters of 4 and 5) is computing risk (and its gradient) using these semirings. It remains to explore more applications that are feasible now by using these semirings. For example, the second-order expectation semiring allows us to compute the covariance matrix of the features on a hypergraph, enabling us to capture the feature interaction during discriminative training. This might help to improve the state of the art of MT performance since feature interaction (e.g., the *interaction* between an n-gram model and a word penalty feature) is largely ignored now. Also, since the computation of many information theoretical quantities (e.g., entropy, cross-entropy, and KL divergence) and their gradients over a hypergraph is feasible now by using these semirings, one may investigate augmenting many current training/decoding objectives with these quantities. This might lead to new and theoretical sounding training/decoding methods for MT.

In chapters 4 and 5, we have presented two unsupervised training methods, which can exploit monolingual English data for discriminative training. The advantage of our methods over a *generative* language model (e.g., an n-gram model) is that they can easily incorporate features into the log-linear model. Compared with *supervised* discriminative training, our methods have the advantage that they can use monolingual data (instead of bilingual data) for training. In this dissertation, we have built a good understanding and a solid infrastructure (within **Joshua**) for these unsupervised training methods. However, we have only performed experiments on a relatively small task (i.e., the IWSLT task) with a relatively small number of features (i.e., about 500), due to computational limitations. It remains to apply our method to a large-scale task (e.g., the NIST Chinese-to-English task) and a task with relatively few bilingual resources (e.g., the NIST translation task between Urdu and English). It may also be worth exploring more novel features in the training.

In Chapter 6, we have successfully applied the variational inference framework to a large-scale MT task, to approximate the intractable problem of MAP decoding in the presence of spurious ambiguity. Many interesting research directions remain open. For example, one may also attempt to minimize $\mathrm{KL}(q \,\|\, p)$ rather than $\mathrm{KL}(p \,\|\, q)$ (see Section 6.1 on page 89), in order to approximate the *mode* (which may be preferable since we care most about the 1-best translation under p) rather than the *mean* of p (Minka, 2005). Variational inference is a general framework for approximate inference and thus it can be applied to many other intractable (or computationally expensive) problems in MT. For example, one may apply it to approximate the expensive intersection between an SCFG and an n-gram model (see Section 2.2.5 on page 21). One may also be interested in applying our method for tasks beyond MT. For example, one may use it to approximate the intractable MAP decoding inherent in systems using HMMs (e.g. speech recognition).

For Future Reference

The latest version of this dissertation, including the inevitable errata, can be found at http://www.cs.jhu.edu/~zfli/. This will also serve as the home for software implementing the algorithms described herein.

Bibliography

Cyril Allauzen, Mehryar Mohri, and Brian Roark. Generalized algorithms for constructing statistical language models. In *ACL*, pages 40–47, 2003. URL http://www.aclweb.org/anthology/P03-1006.

Abhishek Arun, Chris Dyer, Barry Haddow, Phil Blunsom, Adam Lopez, and Philipp Koehn. Monte carlo inference and maximization for phrase-based translation. In *CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 102–110, Morristown, NJ, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-29-9.

J. K. Baker. Trainable grammars for speech recognition. In Jared J. Wolf and Dennis H. Klatt, editors, *Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America*, MIT, Cambridge, MA, June 1979.

Colin Bannard and Chris Callison-Burch. Paraphrasing with bilingual parallel corpora. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 597–604, Morristown, NJ, USA, 2005. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1219840.1219914.

L. E. Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of markov processes. In *Inequalities*, number 3, pages 1–8, 1972.

Christopher M. Bishop. Pattern recognition and machine learning. Springer, 2006.

Phil Blunsom, Trevor Cohn, and Miles Osborne. A discriminative latent variable model for statistical machine translation. In *ACL*, pages 200–208, 2008.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19(2):263–311, 1993. ISSN 0891-2017.

Chris Callison-Burch, Philipp Koehn, and Miles Osborne. Improved statistical machine translation using paraphrases. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Com-*

putational Linguistics, pages 17–24, Morristown, NJ, USA, 2006a. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220835.1220838.

Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluating the role of bleu in machine translation research. In *In EACL*, pages 249–256, 2006b.

Daniel Cer, Dan Jurafsky, and Christopher D. Manning. Regularization and search for minimum error rate training. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 26–34, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W08/W08-0304.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical report, 1998.

David Chiang. A hierarchical phrase-based model for statistical machine translation. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270, Morristown, NJ, USA, 2005. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1219840.1219873.

David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2): 201–228, 2007. ISSN 0891-2017. doi: http://dx.doi.org/10.1162/coli.2007.33.2.201.

David Chiang, Yuval Marton, and Philip Resnik. Online large-margin training of syntactic and structural translation features. In *Proceedings of EMNLP*, 2008.

David Chiang, Kevin Knight, and Wei Wang. 11,001 new features for statistical machine translation. In *NAACL*, pages 218–226, 2009.

W. K. Clifford. Preliminary sketch of bi-quaternions. *Proceedings of the London Mathematical Society*, 4:381–395, 1873.

Michael Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 1–8, Morristown, NJ, USA, 2002. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1118693.1118694.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, 2006. ISSN 1532-4435.

John DeNero, David Chiang, and Kevin Knight. Fast consensus decoding over translation forests. In *ACL-IJCNLP*, 2009.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, number 1, pages 267–271, 1959.

Markus Dreyer and Jason Eisner. Graphical models over multiple strings. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 101–110, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D09/D09-1011.

Markus Dreyer, Keith Hall, and Sanjeev Khudanpur. Comparing reordering constraints for smt using efficient bleu oracle computation. In *Proceedings of SSST, NAACL-HLT 2007 / AMTA Workshop on Syntax and Structure in Statistical Translation*, pages 103–110, Rochester, New York, April 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W07/W07-0414.

Christopher Dyer, Smaranda Muresan, and Philip Resnik. Generalizing word lattice translation. In *Proceedings of ACL-08: HLT*, pages 1012–1020, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P08/P08-1115.

Matthias Eck and Chiori Hori. Overview of the iwslt 2005 evaluation campaign. In *In Proc. of the International Workshop on Spoken Language Translation*, 2005.

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *ACL*, pages 1–8, 2002.

Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *ACL*, pages 205–208, 2003.

Jason Eisner, Eric Goldlust, and Noah A. Smith. Compiling comp ling: practical weighted dynamic programming and the dyna language. In *HLT/EMNLP*, pages 281–290, 2005.

Pedro F. Felzenszwalb and David McAllester. The generalized a* architecture. *J. Artif. Int. Res.*, 29(1):153–190, 2007. ISSN 1076-9757.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. Scalable inference and training of context-rich syntactic translation models. In *ACL*, pages 961–968, 2006.

Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, 1993. ISSN 0166-218X. doi: http://dx.doi.org/10.1016/0166-218X(93)90045-P.

Kevin Gimpel and Noah A. Smith. Cube summing, approximate inference with non-local features, and dynamic programming without semirings. In *EACL*, pages 318–326, 2009.

Joshua Goodman. Efficient algorithms for parsing the DOP model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 143–152, 1996.

Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, 1999. ISSN 0891-2017.

Y Grandvalet and Y Bengio. Semi-supervised learning by entropy minimization. In *NIPS*, pages 529–536, 2004.

Nilsson N. Hart, P. and B. Raphael. A formal basis for the heuristic determination of minimal cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, number 2, pages 100–107, 1968.

R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. Macdonald. Hierarchical a*: Searching abstraction hierarchies efficiently. In *In Proceedings of the National Conference on Artificial Intelligence*, pages 530–535, 1996.

Mark Hopkins and Greg Langmead. Cube pruning as heuristic search. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 62–71, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D09/D09-1007.

Jui-Ting Huang, Xiao Li, and Alex Acero. Discriminative training methods for language models using conditional entropy criteria. In *ICASSP*, 2010.

Liang Huang and David Chiang. Better k-best parsing. In *IWPT*, pages 53–64, 2005.

Liang Huang and David Chiang. Forest rescoring: Faster decoding with integrated language models. In *ACL*, pages 144–151, 2007.

Feng Jiao, Shaojun Wang, Chi-Hoon Lee, Russell Greiner, and Dale Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *ACL*, pages 209–216, 2006.

Mark Johnson, Thomas Griffiths, and Sharon Goldwater. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 139–146, Rochester, New York, April 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N/N07/N07-1018.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In *Learning in Graphical Models*. MIT Press, 1999. URL http://research.microsoft.com/en-us/um/people/minka/papers/message-passing/.

Sanjeev Khudanpur and Jun Wu. Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. In *Computer Speech and Language*, number 4, pages 355–372, 2000.

Dan Klein and Christopher D. Manning. A* parsing: fast exact viterbi parse selection. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 40–47, Morristown, NJ, USA, 2003. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1073445.1073461.

Donald Knuth. A generalization of dijkstra's algorithm. In *Information Processing Letters*, number 1, 1977.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010. ISBN 0521874157.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *NAACL*, pages 48–54, 2003.

Shankar Kumar and William Byrne. Minimum bayes-risk decoding for statistical machine translation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 169–176, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

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*, pages 163–171, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P09/P09-1019.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Adaptive language modelling using the maximum entropy principle. In *Proc. ARPA Human Language Technologies Workshop*, pages 81–86, 1993. URL http://www.sls.lcs.mit.edu/raylau/publications.html#HLTW.

Gregor Leusch, Evgeny Matusov, and Hermann Ney. Complexity of finding the BLEU-optimal hypothesis in a confusion network. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 839–847, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D08-1088.

Zhifei Li and Jason Eisner. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 40–51, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D09/D09-1005.

Zhifei Li and Sanjeev Khudanpur. A scalable decoder for parsing-based machine translation with equivalent language model state maintenance. In *ACL SSST*, pages 10–18, 2008a.

Zhifei Li and Sanjeev Khudanpur. Large-scale discriminative n-gram language models for statistical machine translation. In *AMTA*, pages 133–142, 2008b.

Zhifei Li and Sanjeev Khudanpur. Efficient extraction of oracle-best translations from hypergraphs. In *Proceedings of NAACL*, 2009a.

Zhifei Li and Sanjeev Khudanpur. Forest reranking for machine translation with the perceptron algorithm. In *GALE book chapter on "MT From Text"*, 2009b.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren Thornton, Jonathan Weese, and Omar. Zaidan. Joshua: An open source toolkit for parsing-based machine translation. In *WMT09*, pages 26–30, 2009a.

Zhifei Li, Jason Eisner, and Sanjeev Khudanpur. Variational decoding for statistical machine translation. In *ACL*, 2009b.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 761–768, Morristown, NJ, USA, 2006. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220175.1220271.

R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. J. Wiley & Sons, New York, 1987.

Yang Liu, Qun Liu, and Shouxun Lin. Tree-to-string alignment template for statistical machine translation. In *ACL*, pages 609–616, 2006.

Adam Lopez. Hierarchical phrase-based translation with suffix arrays. In *EMNLP-CoNLL*, pages 976–985, 2007.

Adam Lopez. Translation as weighted deduction. In *EACL*, pages 532–540, 2009.

Wolfgang Macherey, Franz Och, Ignacio Thayer, and Jakob Uszkoreit. Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 725–734, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D08-1076.

Nitin Madnani, Necip Fazil Ayan, Philip Resnik, and Bonnie J. Dorr. Using paraphrases for parameter tuning in statistical machine translation. In *Proceedings of the Workshop on Statistical Machine Translation*, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In *ACL*, pages 75–82, 2005.

Jonathan May and Kevin Knight. A better n-best list: practical determinization of weighted finite tree automata. In *NAACL*, pages 351–358, 2006.

Thomas Minka. Empirical risk minimization is an incomplete inductive principle. In *MIT Media Lab note*, 2000.

Tom Minka. Divergence measures and message passing. In *Microsoft Research Technical Report (MSR-TR-2005-173)*. Microsoft Research, 2005. URL http://research.microsoft.com/en-us/um/people/minka/papers/message-passing/.

Mehryar Mohri and Michael Riley. An efficient algorithm for the n-best-strings problem. In *In Proceedings of the International Conference on Spoken Language Processing 2002 (ICSLP'02*, 2002.

Mark-Jan Nederhof. Weighted deductive parsing and knuth's algorithm. *Comput. Linguist.*, 29(1):135–143, 2003. ISSN 0891-2017. doi: http://dx.doi.org/10.1162/089120103321337467.

Franz Josef Och. Minimum error rate training in statistical machine translation. In *ACL*, pages 160–167, 2003.

Franz Josef Och and Hermann Ney. Improved statistical alignment models. In *ACL*, pages 440–447, 2000.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2001.

Adam Pauls and Dan Klein. K-best a* 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*, pages 958–966, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P09/P09-1108.

B. A. Pearlmutter and J. M. Siskind. Lazy multivariate higher-order forward-mode ad. In *Proceedings of the 34th Annual Symposium on Principles of Programming Languages (POPL)*, pages 155–160, 2007.

Fernando C. N. Pereira and David H. D. Warren. Parsing as deduction. In *ACL*, pages 137–144, 1983.

Slav Petrov, Aria Haghighi, and Dan Klein. Coarse-to-fine syntactic machine translation using language projections. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 108–116, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. pages 185–208, 1999.

Chris Quirk, Chris Brockett, and William Dolan. Monolingual machine translation for paraphrase generation. In *In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 142–149, 2004.

Chris Quirk, Arul Menezes, and Colin Cherry. Dependency treelet translation: syntactically informed phrasal smt. In *ACL*, pages 271–279, 2005.

Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 47–54, Barcelona, Spain, July 2004.

Kenneth Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. In *Proceedings of the IEEE*, pages 2210–2239, 1998.

Roni Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. In *Computer Speech and Language*, number 3, pages 187–228, 1996.

Roni Rosenfeld, Stanley F. Chen, and Xiaojin Zhu. Whole-sentence exponential language models: a vehicle for linguistic-statistical integration. *Computers Speech and Language*, 15(1), 2001.

D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. J. Wiley & Sons, New York, 1987.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1994.

Khalil Sima'an. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *ACL*, pages 1175–1180, 1996.

David Smith and Jason Eisner. Dependency parsing by belief propagation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 145–156, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D08-1016.

David A. Smith and Jason Eisner. Minimum risk annealing for training log-linear models. In *ACL*, pages 787–794, 2006.

Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the Association for Computational Linguistics (ACL 2005)*, Ann Arbor, Michigan, 2005.

Andreas Stolcke. SRILM—an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904, 2002.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: a large margin approach. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 896–903, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: http://doi.acm.org/10.1145/1102351.1102464.

Roy Tromble, Shankar Kumar, Franz Och, and Wolfgang Macherey. Lattice minimum-Bayes-risk decoding for statistical machine translation. In *EMNLP*, pages 620–629, 2008.

AJ Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory*, number 2, pages 260–269, 1967.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. Online large-margin training for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 764–773, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D07/D07-1080.

Richard Zens and Hermann Ney. N-gram posterior probabilities for statistical machine translation. In *WMT06*, pages 72–77, 2006.

Hao Zhang and Daniel Gildea. Efficient multi-pass decoding for synchronous context free grammars. In *ACL*, pages 209–217, 2008.

Vita

Zhifei Li was born in Hunan province of China, in August, 1976. He obtained a B.S. degree (majored in Thermal Engineering) from Nanjing University of Science and Technology in 1999, and a master degree (majored in Computer Science) from Nanjing University of Aeronautics and Astronautics in 2002.

Zhifei is currently a Ph.D. candidate in Johns Hopkins University's Computer Science Department and Center for Language and Speech Processing (CLSP). His research interests are in machine translation, speech recognition, machine learning, applied algorithms, and wireless networks. He is the primary creator of Joshua, an open-source toolkit for parsing-based machine translation. His work was nominated for a best paper award in ACL 2009. He was also a finalist for a Microsoft Research Ph.D. fellowship.