Latent Structure Discriminative Learning for Natural Language Processing

by

Ann Clifton

M.Sc., Simon Fraser University, 2010
B.A., Reed College, 2001

Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy

in the
Department of Computing Science
Faculty of Applied Science

Approval

Name:	Ann Clifton
Degree:	Doctor of Philosophy (Computing Science)
Title:	*Latent Structure Discriminative Learning for Natural Language Processing*
Examining Committee:	**Dr. Ted Kirkpatrick** (chair) Professor

Dr. Anoop Sarkar
Senior Supervisor
Professor

Dr. Fred Popowich
Co-Supervisor
Professor

Dr. Greg Mori
Supervisor
Professor

Dr. Marine Carpuat
External Examiner
Assistant Professor
Department of Computer Science
University of Maryland

Date Defended:	23 October 2015

Abstract

Natural language is rich with layers of implicit structure, and previous research has shown that we can take advantage of this structure to make more accurate models. Most attempts to utilize forms of implicit natural language structure for natural language processing tasks have assumed a pre-defined structural analysis before training the task-specific model. However, rather than fixing the latent structure, we may wish to discover the latent structure that is most useful via feedback from an extrinsic task. The focus of this talk is on jointly learning the best latent analysis along with the model for the NLP task we are interested in.

In this work, we present a generalized learning framework for discriminative training over jointly learned latent structures, and apply this to several NLP tasks. We develop a high-accuracy discriminative language model over shallow parse structures; in addition, we set forth a framework for latent structure learning for machine translation, in which the latent segmentation and alignment of the parallel training data inform the translation model.

Keywords: discriminative learning; latent variable models; structured learning; statistical machine translation; language modeling

Dedication

To my family!

Acknowledgements

It is my pleasure to thank my parents, Anoop, Fred, Greg, my lab colleagues, and my family and friends.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Summary

In this thesis we will explore learning latent structure in discriminative models for Natural Language Processing (NLP). There are many different layers of implicit structure in natural language corresponding to different levels of analysis, e.g., phonological, morphological, syntactic, semantic, and this thesis explores how we can better use this hidden information for NLP tasks.

There has been previous research showing that using latent structure can help model performance, for example, learning the grammatical structure of text data is a useful step in many NLP tasks, such as semantic parsing and question answering. However, the bulk of this work has treated the step in which we infer the hidden structure as something of a black box, using human-generated structural annotations where possible and automatic methods elsewhere. The inferred latent structure is then generally taken to be fixed when used within some extrinsic task.

The problems with this approach are that human-generated structural annotations are expensive to get, and the automatically generated annotations are error-prone. So, rather than taking some fixed latent structure from an external source, we would like to learn our own hidden structure model in conjunction with learning a model for some extrinsic task, which will provide feedback to the latent learner. In this way we can allow our task-specific model to benefit from information generated by a latent learner specifically optimized for the task at hand.

We first apply this framework to a discriminative language model (LM), which we frame as a binary grammaticality classification task. In learning to distinguish between positive and negative examples of grammatical text, the model iteratively learns a shallow parsing model which provides structural features to the binary classifier. We extend this model to learning over multiple different representations of the latent structures, and consider different methods of combining the information from the different latent learners. We show that this method attains a high accuracy on a grammaticality classification task, and that it can learn a high quality model more quickly than using a single representation of the latent structural information.

We then investigate how this learning framework can be applied to Statistical Machine Translation (SMT). We again set up the problem as a discriminative learning task, in this case that classifies the quality of translation output, while learning a model that jointly aligns and segments the parallel input data. Data segmentation can be an important step in producing good quality SMT output, particularly for languages such as Chinese for which no word-based segmentation is necessarily present, and for morphologically rich languages, in which the word-based segmentation may too densely packed with information to get useful units for translation to or from a more isolating language.The segmentation-alignment model is guided by feedback from the external loss over the translation quality, and is thus able to optimize the segmentation of the data for the translation task. We focus on an English-Turkish translation task, and we find that by allowing our model to consider sub-word information in making segmentation and translation decisions, it is able to improve performance over both a word-based and a morphologically segmented model.

For all the models and tasks described in this thesis, we would like to perform training on large data sets when possible, although the large number of parameters make this conputationally expensive to do. However, since these models lend themselves easily to parallelization, we also examine empirical considerations for how to train these models in the distributed setting in order to make the best use of large data with finite computational resources. We consider computational topologies for performing large-scale distributed training for these kinds of structured models, and find that the choice of distributed representation of the data and topology of the learners can affect both the speed and accuracy of model training.

1.2 Introduction

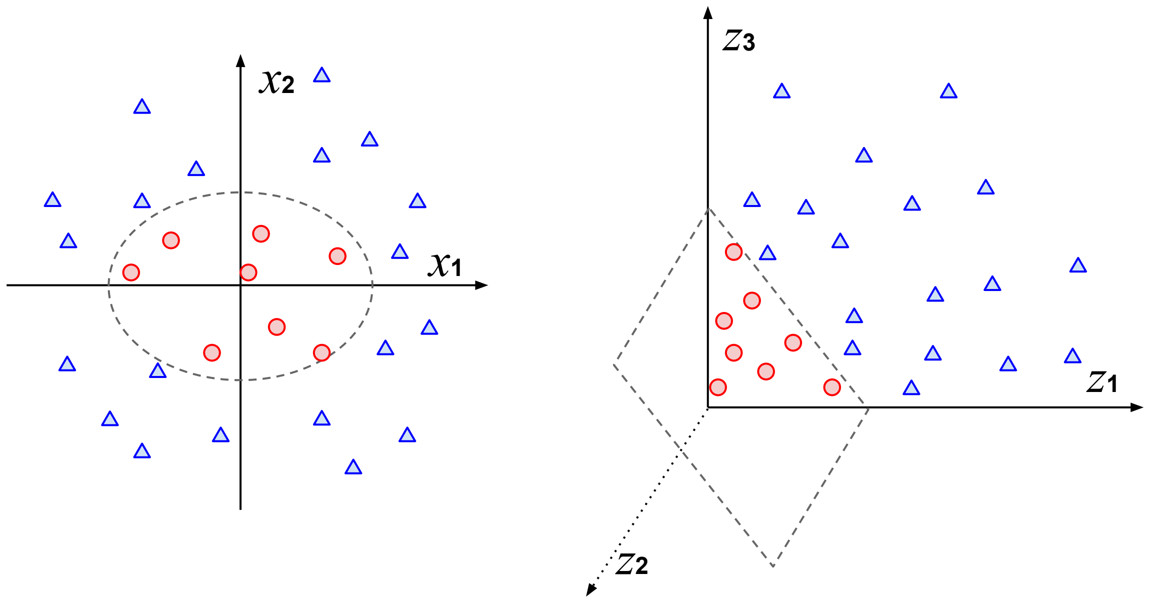
Natural language is rich with ambiguity. This can come in the form of lexical ambiguity, when a word has multiple meanings (e.g. 'bark': from a tree, or from a dog), or in the form of syntactic ambiguity ('I helped the child with the raft'). This is what makes computational Natural Language Processing (NLP) difficult. Linguists believe that humans are able to resolve all this ambiguity so deftly by drawing upon external information other than just the speech signal or printed text, such as conversational context cues or knowledge about the world. In addition, they also make use of some internal assumptions about the way language data is structured in order to make sense of the words on the page or the speech signal.

It is these inferred multiple layers of hidden structure we are interested in recruiting for NLP. Consider, for example, how principled patterns of phonology govern word pronunciation in context, and patterns of syntactic well-formedness govern sentence structure. We can even extend this view to larger linguistic units than the sentence– there has been considerable study of the discursive structures that emerge from document-level natural language data. Given the high level of ambiguity in raw text or speech data, it is therefore natural to investigate how these kinds of hidden structures can be used to help with NLP tasks. For example, suppose we are performing a language-understanding task and we are given the sentence "felons appeal to the court." In this case, it would be very helpful to know the semantic structure of the sentence in order to distinguish between whether this sentence describes the predilections of the court or an action taken by felons.

When using learning latent structure in order to inform some other task, we may consider this a form of semi-supervised learning, in which we are provided with labels for the extrinsic task, but not for the latent structure. There has been significant work done in the area (see [46, 55]), however, the majority of this has been within the generative setting; we will discuss this further in Chapter 3. There has been much less work done of this flavor in a discriminative framework; in this thesis we will explore learning latent structures to inform discriminative models.

1.3 Discriminative Learning with Latent Structures

For a variety of tasks, it is often the case that we wish to learn the classification boundary between two or more types of observed data points. However, in many domains, and particularly when dealing with natural language data, there may not exist a boundary that neatly separates all points of each class from each other.



The idea behind latent discriminative learning is that if we project our observed data points into some higher-dimensional feature space, we can find the separating surface between our data classes. This general intuition fits nicely with our linguistic knowledge of natural language data, which we know to be structured on many different levels. These structures are not directly observed but rather inferred from the observed language data. We can think of these latent structures as a source of information to predict groupings of and relations between observed data points. Thus, for a simple example, if we want to decide whether 'bat' should be clustered with the word 'rodent' or the word 'hit', it will help us to know the more general, but unobserved part-of-speech (POS) of 'bat'.

Previous work has used information from inferred latent structures to inform classifiers, e.g. [5], [18], [24], and [33]. However the limitation of this approach is twofold. First, the methods used to generate these latent structures are rarely foolproof, and thus run the risk of propagating error from the latent structures found into the classification pipeline. Additionally, generating latent structure (say, getting a parse for a sentence) is quite expensive to do by hand, and even hand-crafted structural annotations may not be agreed upon by all annotators; in many tasks there are multiple right answers (consider, for example, translation). Basically, the problem in using a pipelined approach towards using inferred structures in classification comes down to the abundance of ambiguity in natural language. However, rather than viewing this to be a disadvantage, by learning over latent structures, we can allow the task to guide the selection of the latent structure implicit in the data. To consider it another way, we may not in fact prefer to learn the gold-standard, linguistically correct hidden structure for the observed data. Since the NLP tasks we are interested in may not exactly replicate the output of human cognition, we way instead prefer to learn the latent structures that will help us most with the actual task as we have defined it, and let the desired outcomes of our task guide the model learned for the latent information.

1.4 The Tasks

1.4.1 Discriminative Language Modeling with Latent Syntax

The first task that we explore is a language modeling task, which we frame as a binary grammaticality classifier that uses feedback from a shallow parsing model in judging the quality of text. Let us consider an example to motivate this approach: suppose we want to judge the quality of the following two sentences:

(1) 'The golden brown dog slept.'
'The golden brown over medium.'

The most common way of doing this would be to use an n-gram LM, which estimates the quality of a piece of text based on looking at the windows of length n and checking how often they have been seen before in the training data. So, if we were using a trigram

LM, we would judge the first sentence on how frequently we have previously seen (The golden brown), (golden brown dog), and (brown dog slept). The second sentence would be judged based on the frequency of (The golden brown), (golden brown over), and (brown over medium). All of these are reasonably likely trigrams, so our trigram LM would have a hard time judging one to be much better than the other, though a human English speaker could easily tell you that the first sentence is permissible but the second is definitely not.

However, if we know something about the latent grammatical structure of these two sentences, it clarifies things. Without attempting to generate a full parse tree, just chunking the sentence into its constituents using part-of-speech (POS) tags helps distinguish between these two sentences. For the first sentence, it is straightforward to assign it a sequence of POS tags and constituent chunking:

(2)	The/DET	golden/ADJ	brown/ADJ	dog/NOUN	slept/VERB.
	Begin-NP	Inside-NP	Inside-NP	Inside-NP	End-NP

Since the second sentence is ungrammatical, it is more difficult to come up with a definitive POS sequence and chunking, but a couple of possible such sequences are:

(3)	The/DET	golden/ADJ	brown/ADJ	over/PREP	medium/ADJ
	Begin-NP	Inside-NP	End-NP	Begin-PP	Inside-PP
	The/DET	golden/ADJ	brown/VERB	over/PREP	medium/ADJ
	Begin-NP	End-NP	Begin-VP	Inside-VP	End-VP

In either case, without some serious effort, we are left with a chunking sequence that includes a noun phrase constituent that lacks a noun, or some other structure that is equally improbable to be produced in natural language. The idea is that while the words and n-grams in these two sentences may be similarly probable, their latent structures will be different enough to allow our model to differentiate between them more easily. So, we can extract features from hidden structures that our model infers to inform the classification task, and also allow feedback from that task to influence the way that our model chooses the latent structures.

Previous work on leveraging latent representations for discriminative language models has used batch algorithms that require multiple passes though the entire training data. Instead, we propose an online algorithm that efficiently jointly learns the latent structures and the classifier. We further extend this to include multiple views on the latent structures with different representations, and we explore a variety of ways to incorporate these multiple views in learning the classifier and their interaction. We evaluate our algorithms in comparison to the most closely related batch algorithm, Learning over Constrained Latent Representations (LCLR), and show that our online algorithm significantly outperforms it on a grammaticality task. This completed work is described in [14].

1.4.2 Joint Latent Segmentation and Alignment for Discriminative Machine Translation

We next investigate applying this discriminative learning framework to SMT over latent segmentation and alignments. The key idea behind this work is that languages encode information differently, and so for the purposes of translation, we want to take advantage of similarities and try to bring into greater symmetry areas of great difference. This is particularly important for translation for morphologically rich languages. Consider, for example, the following sentence from Turkish:

(4)

çöp	+lük	+ler	+imiz	+de	+ki	+ler	+den	+mi	+y	+di
garbage	+AFF	+PL	+1p/PL	+LOC	+REL	+PL	+ABL	+INT	+AUX	+PAST

‘Was it from those that were in our garbage cans?’

In English, we require ten separate words to render the equivalent meaning of the one-word Turkish sentence. So, if we are dependent on the word-based segmentation such as this to train our translation system, our model will have to memorize sentence-long chunks of English in order to capture Turkish words such as this one effectively.

In general, SMT systems tend to perform better on language pairs that are closely related or have a high degree of symmetry. This is due to many reasons (for example, less reordering is required), but an important factor has to do with the granularity of the language and the way it encodes information. So if I wish to translate English into Turkish, I may wish to break up the Turkish side into chunks that more closely resemble English. Consider a much simpler example from Spanish; suppose I have the following two sentences in my training data:

(5)

no	want-1SG	bring-INF-it-OBJ
No	quiero	traerlo.

‘I don’t want to bring it’

(6)

what	go-2SG	to	bring-INF
Qué	vas	a	traer?

‘What are you going to bring?’

and we need to translate the following new sentence:

(7)

it-OBJ	have-2SG
Lo	tienes

‘You have it’

In this case it will be helpful to consider breaking up the Spanish side of the parallel data to help the translation task. Given that we have seen 'traer' and 'traerlo' in our training data, if our model is able to consider sub-word segments of the data, it may be able to learn that 'traerlo' can be split into 'traer' and 'lo,' and thus be able to suggest a translation for 'lo' when it encounters it later. However, we may not want to split up every morphologically inflected word form into its component parts– for example, we may have seen 'va' translated as 'she goes' and 'vas' translated as 'you go' enough times that there is no benefit in analyzing 'vas' into its morphological components.

This is the main idea behind our approach to discriminative SMT: we allow the model to consider possible latent segmentations of the data in alignment, and we optimize those segmentations for the translation task, guiding the joint model towards the segment-alignments that produce the best translations

We implement and evaluate end-to-end discriminative max-margin training for machine translation where the alignment and the segmentation of the source sentences are latent variables. Our alignment model differs from much work in discriminative training for MT in that it is end-to-end rather than doing only post-processing on the output of a generative model. It also differs from some other work in discriminative end-to-end models in modeling the both the alignment as well as the source-side segmentation as latent variables and in using a max-margin optimization. We apply this model towards an English-Turkish translation task. We find that using the alignments learned by the translation-segment driven model improve performance over a phrase-based baseline.

1.5 Contributions

Previous work has considered using latent structures for NLP. The main contributions of this thesis can be summed up as follows.

First, in our approach towards building a discriminative language model, our work uses multiple representations of latent structures, as opposed to a single representation for the latent information. In supervised structured learning tasks, various forms of model combination and multiple representations have been found to be helpful when learning the structure is the explicit task. This is the first work to allow the model to range over multiple representations of the latent structure to guide the multiple learners towards the most useful features for the explicit task.

The other main contribution is towards discriminative SMT, specifically in how our joint segmentation-alignment model makes use of the latent information. In particular, this work builds upon previous efforts towards discriminative SMT by novelly jointly learning over segmentations as well as alignments. Previous work using latent structures for SMT has been concerned with latent derivations of the output that specify a mapping from a fixed input to the output generated. In contrast, our model makes it possible to learn a latent representation of input, not just of derivations of the output, and therefore to optimize the segmentation the model learns for the translation task.

1.6 Outline

In the remainder of this thesis, we will begin with introductory material that summarizes the basics about the tasks and models that will be used. In Chapter 2 we give background about the tasks we tackle in this thesis: we will begin by discussing canonical approaches towards language modeling in section 2.1; we will then briefly cover typical SMT methods in section 2.2, focusing on phrase-based approaches. Chapter 3 gives an overview of latent variable modeling for NLP, starting broadly and focusing in on the particular models of interest for this thesis. We introduce latent variables and structured models, and then how these can be combined in the discriminative setting. In Chapter 4 we describe our work with discriminative language modeling over latent shallow syntax, where we will give a detailed description of the task, the model and its training, and our experiments. Chapter 5 presents our work on joint discriminative alignment and segmentation for SMT, where the task, the model, and the experiments will be described in more detail. We turn to a consideration of how to select the topology of distributed computation for training these kinds of models in Chapter 7. The conclusion and future work are given in Chapter 8.

Chapter 2

Language Modeling and Statistical Machine Translation

In this chapter we give an overview of the two NLP tasks central to this work. In section 2.1 we briefly introduce language modeling and in section 2.2 we give an introduction to statistical machine translation (SMT), focusing on phrase-based MT.

2.1 Language Modeling

Language models are used in NLP to assign likelihood to text, based on its similarity to a set of language data used to train the model. Language models were originally developed primarily for the task of speech recognition, but are widely used in a variety of applications. For example, language modeling is a key component of Machine Translation, in which it supplies judgements on the fluency of the MT output.

Language models are most often formulated in a generative framework, in which we use a markov model over fixed-length sequences of tokens or n-grams. These models assign a probability to a piece of text given its history, and are estimated by counting up the frequency of the different histories that tokens appear with in the training data. The tokens over which these models are estimated are generally taken to be words, but can be estimated over larger units such as phrases or over smaller units such as morphemes, phonemes, or characters.

In this framework, the probability of a sequence such as a sentence is given as the product of the probabilities of the tokens in the sequence. So, for sequence S of length L composed of tokens $s_0..s_L$, the probability of the sequence is given by

$$P(S) = \prod_{i=0}^{L} p(s_i). \tag{2.1}$$

The probability of each token s_i is given by the maximum likelihood estimate of s_i given the history $s_{i-1}..s_{i-n}$, i.e., the normalized counts of how often we see token s_i following the sequence $s_{i-1}..s_{i-n}$ versus all other possible token that can follow this sequence.

N-grams LMs are simple and easy to use, and can capture helpful information about the local fluency of a short text sequence. For example, they are easy to integrate into a decoder for machine translation. However, they are limited in their ability to capture long-distance, overlapping, or discourse-level information about text that we may wish to use. For this reason, efforts have been made towards developing other approaches toward language modeling that are able to leverage these kinds of information, including Neural Netwok based and discriminative LMs, which will be discussed further in 4.

2.2 Statistical Machine Translation

The current state of the art SMT systems use generative word-alignment models and log-linear phrase-based translation models, so we begin with a brief introduction to these underlying models.

2.2.1 Generative Models: Word Alignment

The basic SMT model starts by learning word-by-word alignments between each parallel sentence in the corpora.

In a general translation model, we are looking for the target sentence t that is the best translation of source sentence s, i.e., that maximizes $\Pr(t|s)$:

$$\hat{t} = \arg\max_t \Pr(t|s), \tag{2.2}$$

which, by Bayes' rule, we can reformulate as:

$$\hat{t} = \arg\max_t \Pr(t) \Pr(s|t). \tag{2.3}$$

The second term is the probability of the target sentence, which can be supplied by the language model. The first term, $\Pr(t)$, is just the likelihood of sentence t occurring in the target language on its own, so then to find the translation probability $\Pr(s|t)$, we use word alignments between the source and target sentences. For example, Figure 2.1 shows the aligned words of an English-French parallel sentence.

Between any given source sentence $s_1^J = s_1, .., s_J$ and target sentence $t_1^I = t_1, .., t_I$, there may be many possible alignments, some more probable than others; we denote the probability of any particular alignment a as $\Pr(a|s_1^J, t_1^I)$. Since the probability of translating sentence s_1^J as sentence t_1^I is equal to the sum over all possible alignments between them, we can use this in conjunction with the Bayes rule to define $\Pr(s_1^J|t_1^I)$ in terms of alignments:

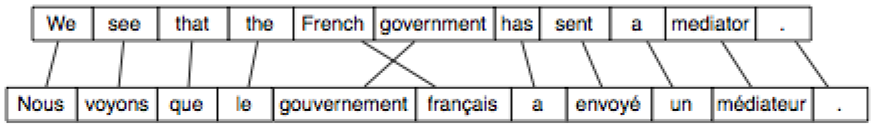


Figure 2.1: Unidirectional Alignment [29]

$$\Pr(s_1^I|t_1^J) = \sum_a \Pr(a, s_1^J|t_1^I), \tag{2.4}$$

which gives us a way to estimate the translation probabilities. It is important to note that while GIZA++ [37], the alignment model used in many SMT systems, allows multiple source words to be aligned to any given target word, it allows only one target word to be aligned to each source word. In order to overcome this limitation, the alignments are calculated in both directions, from source to target, and target to source.

These word-by-word alignments are learned using the expectation-maximization (EM) algorithm, which starts with an initial estimate of uniform probabilities for the word-to-word translation probabilities, $\Pr(t|s)$. These word translation probabilities are then used to compute the whole-sentence alignment probabilities for each sentence in the corpora, $\Pr(a|s_1^J, t_1^I)$. The sentence alignments are then in turn used to recalculate the word alignments; this process repeats iteratively until convergence.

2.2.2 Log-Linear Models for Phrase-Based Translation

Once the model has the whole-sentence word alignments, it can then extract phrase pairs for building the phrase-based translation model by creating a set of alignment points. To do this, the model uses the bidirectional alignments, starting with the intersection of the two. This set of alignment points on which both alignment directions agree is of high precision and represents a lower bound on the size of the set. From here, the set of alignment points is enlarged according to expansion heuristics, with an upper bound of the union of the two alignment directions. Figure 2.2 shows the bidirectional alignment between a parallel English-French sentence, while 2.3 gives its corresponding alignment matrix.

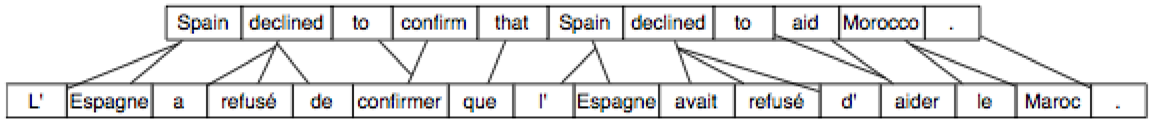
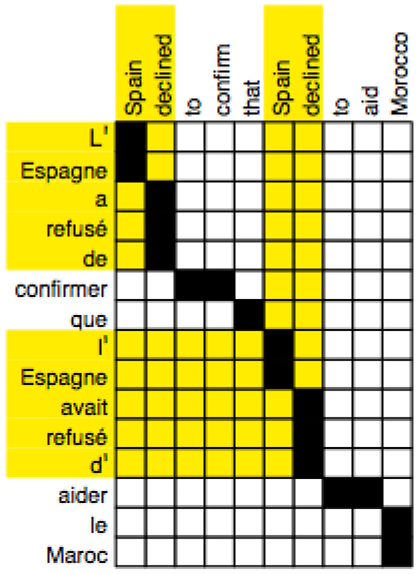


Figure 2.2: Bidirectional Alignment [29]



L' Espagne	Spain
L' Espagne a refusé de	Spain declined
L' Espagne a refusé de confirmer	Spain declined to confirm
a refusé de	declined
a refusé de confirmer	declined to confirm
a refusé de confirmer que	declined to confirm that
confirmer	to confirm
confirmer que	to confirm that
que	that
que l' Espagne	that Spain
que l' Espagne avait refusé d'	that Spain declined
l' Espagne	Spain
l' Espagne avait refusé d'	Spain declined
l' Espagne avait refusé d' aider	Spain declined to aid
...	...

Figure 2.3: Alignment Matrix [29]

The basic expansion heuristic used to build the set of alignment points (from which the phrases will be extracted)starts with an alignment point set from the intersection of the two alignments and then adds neighboring points from the directed alignments not already in the set. In this case, a point's 'neighbors' are defined as those points in the matrix which are vertically, horizontally, or diagonally adjacent; neighboring points from the directed alignments are added to the set until all candidate points have been added.

From this completed set, each legal phrase pair $(\bar{s}, \bar{t})$ that is consistent with the alignment is extracted. The phrases may be a single word, or a group of words, as long as the aligned words in the phrase are only aligned to each other, not any word beyond the phrase boundaries. These phrase pairs can then be used to compute the phrase translation probabilities for each direction, based on their overall counts in the training corpora:

$$\Pr(\bar{s}|\bar{t}) = \frac{\text{count}(\bar{s}, \bar{t})}{\text{count}(\bar{t})}. \tag{2.5}$$

From the phrase pair translation probabilities, we get the sentence translation probabilities, by splitting source sentence s_1^J into phrases $\bar{s}_1^P = \bar{s}_1, .., \bar{s}_P$. The component phrases are translated into P target phrases $\bar{t}_1^P = \bar{t}_1, .., \bar{t}_P$, giving the sentence translation probability as:

$$\Pr(t_1^I|s_1^J) = \prod_{p=1}^{P} \Pr(\bar{t}_p|\bar{s}_p). \tag{2.6}$$

When selecting the best translation output, we would like the model to take into account various other features of the data, in addition to these phrase-based sentence translation probabilities. So in order to make it easier to add other arbitrary feature functions into our model, we take the log of the sentence translation probabilities as well as any other features we wish to add, to combine them in a log-linear model:

$$f_{\tau} = \log \Pr(t_1^I | s_1^J) = \sum_{p=1}^{P} \log \Pr(\bar{t}_p | \bar{s}_p), \tag{2.7}$$

where f_{τ} is the translation feature. We can now add other feature components to this in a linear fashion. By taking the exponent, we can formulate the sum of all of our model features thus:

$$\Pr(t_1^I | s_1^J) = \frac{\exp(\sum_{i=1}{}^{n} \lambda_i f_i(t_1^I, s_1^J))}{\sum_{t'^I_1} \exp(\sum_{i=1}{}^{n} \lambda_i f_i(t'^I_1, s_1^J))} \tag{2.8}$$

where λ_i is the weight parameter for feature f_i.

It is infeasible to enumerate all possible combinations to calculate the normalization constant in the denominator; we therefore use nbest-lists to approximate this in training the model weights. However, since the denominator is a constant, we can ignore this when decoding, finding the best translation for sentence s_1^J using:

$$\hat{t}_1^I = \arg\max_{t_1^I} \; \exp(\sum_{i=1}^{n} \lambda_i f_i(t_1^I, s_1^J)). \tag{2.9}$$

The components we wish to add in addition to the translation probabilities are the distortion model, the word penalty, and the phrase penalty, and the language model. The distortion model captures how much reordering of the target-side phrases should be done; the word penalty regulates the length of the target-side translation output, and the phrase penalty captures how many phrase pairs should be used for each sentence.

2.2.3 Evaluation - BLEU, WER and TER

The evaluation measures used for measuring the quality of a translation warrant discussion. Since it is impractical to have access to human judgement of translation on a quick and regular basis, evaluation techniques were developed to approximate this judgement automatically. The most common standard measure is the BLEU (Bilingual Evaluation Understudy) [40] score, which has been shown to be closely correlated with human judgments of translation quality, and is based on n-gram correspondences between a candidate translation and a reference translation. "N-gram" simply refers to sequences of contiguous items (in our case, words) in a text, where n specifies the number of items. BLEU performs a comparison over the whole corpus based on the number of n-grams from the reference translation that appear correctly in the MT output translation. By factoring in the scores for each order of n-gram, the BLEU method captures both content accuracy and fluency; where unigrams reflect the extent to which the reference captures all the information, longer n-grams reflect fluent, well-constructed sentences. Scoring a candidate translation on the basis of n-grams allows for multiple possible orderings of a translation to be deemed equally valid. This approach was designed with an eye to the fact that there are typically many ways to translate a sentence, and is more robust if multiple references are provided. Thus, in order to make the evaluation compatible with multiple references without requiring a candidate translation to include all the different phrasings of the same input, this measure captures precision but sacrifices recall. The candidate translation's n-gram precision score is modified to make sure that it does not get credit for repeating an n-gram more frequently than it was seen in the reference. This is known as *clipping* the n-gram counts so as not to exceed their maximum in the reference.

BLEU does not explicitly account for recall, however, it can capture recall to some degree when supplied with a large number of reference translations. In order to compensate for this, BLEU includes a brevity penalty to keep the scores of translations that are composed of reference words but are shorter than the reference from scoring artificially highly. To get an intuition for why this is necessary, we include the following example:

(8) Candidate: of the

(9) Reference: It is the practical guide for the army always to heed the directions **of the** party. [40]

In this example, the candidate translation is clearly very poor, but since it is composed of elements found the reference, and these appear the same number of times in the candidate and reference, its unigram precision is $\frac{2}{2}$ ('of' and 'the'), and its bigram precision is $\frac{1}{1}$ ('of the'). To keep this type of short translation from getting a perfect score, then, the brevity penalty is included, which decreases the candidate's score proportionally to how much shorter it is than the reference. The brevity penalty (BP) is computed as:

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r, \\ e^{\frac{1-r}{c}} & \text{if } c \leq r. \end{cases}$$

Then the BLEU score is:

$$\log \mathrm{BLEU} = \min(1 - \frac{r}{c}, 0) + \sum_{n=1}^{N} w_n \log p_n, \tag{2.10}$$

where w_n are the positive weights for each order n-gram, summing to one.

Since BLEU keeps track of the number of reference n-grams that appear in the system output translations, in theory the best score is 100%. In practice, however, scores are much lower. The best systems tend to achieve scores in the high 30% to 40% range. However, this is very strongly affected by the amount of parallel data available between the language pair under consideration, and how closely related the two languages are.

Since BLEU is the current standard measure in most common use and arguably has the highest correlation with human judgement [3], we use this as our primary evaluation tool. However, some studies also report translations' Word Error Rate (WER) [36] and Translation Edit Rate (TER) [32]. These measures are based on the edit-distance between a translation candidate and a reference. WER is an edit-distance measure that looks for the shortest sequence of insertion, deletion, and substitution operations to get from the candidate to a reference. TER also includes the operation of swapping of adjacent phrases, allowing for phrase reordering. These edit-distance based measures are intended to have a high correlation with human judgement with fewer references than BLEU, and penalize phrasal shifts less than BLEU. Unlike BLEU, for WER and TER, lower scores are better. Throughout this work we will report BLEU scores where possible, but we will include WER and TER scores in some cases where BLEU is unavailable.

2.2.4 Summary

In this chapter we have given a brief introduction to the most common methods in language modeling and machine translation. These methods are very prevalent and well-developed but have distinct limitations. These limitations become more pronounced when we try to use these techniques in disadvantaged data settings, particularly for languages that have a high degree of morphological complexity. We will explore other methods that attempt to address these issues in the following chapters.

Chapter 3

Latent Variable Models

In this chapter, we will give an overview of latent variable models, beginning with an introduction to latent variables for NLP, and then considering the case of modeling over latent structures. We will discuss how these can be used particularly within a discriminative learning framework, and consider different approaches to this and their implications for the complexity and convergence properties of training.

3.1 Latent Variable Models

In statistics as well as in many areas of computational modeling, it can be useful to consider latent variables. By latent variables, we mean some property of the data that is not directly observable; we may wish to explain the observed data by positing that each data point also has some attribute or belongs to some class that is hidden. For example, in speech recognition, given an audio recording of multiple people speaking, we may wish to make a guess as to which person is speaking at a given time in order to make a better prediction about what they said. Latent variable models are used widely in NLP, for tasks such as part-of-speech tagging, coreference resolution, and PCFG annotation.

A well-known example of latent variable modeling in NLP is that of topic modeling. Topic models assume that within natural language documents there exist one or more topics, and that knowing the topic(s) of a piece of text can help to characterize and interpret it. For example, having topic information can help in the search for documents that might be relevant to a query, or they can help a translation model choose between multiple possible translations of a word. However, this kind of information is rarely explicitly labeled; even in the case when topic annotations are present they may not be at the granularity that is most useful.

Approaches to topic modeling have been developed with increasing sophistication, but here we will describe a simple Latent Dirichlet Allocation (LDA) topic model to provide an introductory example to latent variable models. This approach to topic modeling proceeds in a generative fashion, where we assume that documents are composed of a mixture of latent topics, and that each of these topics has different preferences for different words. For example, we may consider a sociolinguistics text to contain several different topics, such as linguistics, sociology, pragmatics, etc; and that a word such as ‘syntax’ may be more favored by the linguistics topic than by sociology. In LDA, the basic generative story for a document amounts to first picking a distribution over topics, and then for each word position in the document, picking a topic from the topics distribution, and then picking a word from that topic. To learn the model, the task is to infer the distribution over topics in the document and over words in each topic. Given the observed words in a document, our goal is to estimate the hidden topics that generated them. By modeling the joint probability distribution between the observed words and the hidden topics, we can compute the conditional distribution of the latent topic variables.

More formally, we define the joint distribution over the hidden topics and observed data (as a collection of documents $d_{1..D}$, each composed of words $w_{1..N}$) assuming a fixed vocabulary of words and assuming a fixed number of topics $\beta_{1..K}$. Each topic β_k denotes a distribution over words in the vocabulary; to characterize how prevalent a topic is in a document, we use the parameter $\theta_{d,k}$ to denote the the proportion of a topic k in a document d. To link this to the observed variables, we use topic assignments $z_{d,n}$ for each word w_n in d, which depend on the document-specific topic proportions; according to our generative story, the words $w_{d,n}$ depend on the topic assignments and the distribution over topics:

$$p(\beta_{1..K}, \theta_{1..D}, z_{1..D}, w_{1..N}) = \prod_{K} p(\beta_k) \prod_{D} p(\theta_d) (\prod_{N} p(z_{d,n}|\theta_{d,k}) p(w_{d,n}|\beta_k, z_{d,n})). \tag{3.1}$$

Given this formulation, we wish to compute the conditional distribution over topics given the observed data. In order to do this, we want to marginalize over the hidden topic variables in equation 6.6. It’s easy to compute the joint distribution given any particular setting of the hidden variables, but getting the conditional distribution requires summing over all their possible settings, which is intractable. Thus, this is generally estimated approximately, using either sampling or variational methods.

Latent variable models can be used for unsupervised learning, wherein we directly try to learn the hidden values, as in the case of topic modeling, but they can also be used in supervised learning, wherein we infer the values of the hidden variables in order to help with some explicitly measurable extrinsic tasks. We will discuss this type of latent variable modeling in Chapters 4 and 5.

3.2 Structured Models

In section 3.1, we discussed latent variable models, focusing on the task of topic modeling. However, there are many problems where rather than assigning a single label or distributions over labels for some data, we would like instead to predict a whole latent sequence of labels over the data, in which each component in the label sequence may take into account overlapping views on the data, or the labels themselves may be interrelated. This thesis focuses on this type of learning over sequences, or structured models. So, to give a brief overview to structured models, we use the task of part-of-speech (POS) tagging as an example.

The POS-tagging task assumes that we have an input sequence x composed of words $S = x_1..x_n$, and we wish to find the tag sequence y which for each x_i gives the correct POS tag y_i denoting the grammatical category of x_i. For example, for the input sentence

'While winter reigns the earth reposes'

the corresponding tagged output would be:

'While/ADV winter/NOUN reigns/VERB the/DETERMINER earth/NOUN reposes/VERB'.

This is a very common NLP task, and is an important component of many other tasks such as grammatical and semantic parsing. The main challenge in POS-tagging is the ambiguous meaning of word forms: the same string of characters can denote multiple meanings with different grammatical categories, e.g., 'bat' (the winged mammal) and 'bat' (to hit). The word tokens in isolation may be informative– it might be much more frequent for the verb form of 'bat' to occur in text than the noun form– but this is usually not enough to make high-accuracy tagging predictions for ambiguous words. Looking at the context in which the token occurs can help resolve this ambiguity (e.g. we can make a good guess that if 'bat' follows the word 'the' it is more likely to be a noun than a verb). So, when predicting a label for any point in a sequence, we would like our structured model to take in to account more than just the corresponding point in the sequence; we would also like for it to use aspects of the context surrounding the point in question.

To learn a structured model that takes both the lexical preferences of the individual tokens as well as the surroundings, we can use a generative model over (x, y) pairs:

$$p(x, y), \tag{3.2}$$

which, using Bayes' rule, can be decomposed into:

$$p(y)p(x|y), \tag{3.3}$$

which allows us to model separately the likelihood of the label y and the likelihood of seeing observation x given that that we have already seen label y, known as the noisy channel model.

Using this model, we can also derive a function to give us an estimate for the best label y out of all possible labels Y for a new test example x:

$$f(x) = \arg\max_{y \in Y} p(y)p(x|y), \tag{3.4}$$

A standard way of computing the best sequence in Equation 3.2 is to use a Hidden Markov Model (HMM). A HMM consists of a set of a set of labels Y, observable tokens X, transition probabilities $p(y_i|y_{i-1}..y_{i-m})$ that specify the likelihood of passing from one label to the next (given some amount of history of length m of the previous labels in the sequence), and emission probabilities $p(x_i|y_i)$ that specify the likelihood of seeing an observation given a particular label. With this we can define the probability of a sequence of length n as follows:

$$p(x_0..x_n, y_0..y_n) = \prod_N p(y_i|y_{i-1}..y_{i-m}) \prod_N p(x_i|y_i) \tag{3.5}$$

In a HMM we assume that the label history we consider when computing $p(x_i)$ and $p(y_i)$ is limited to some finite window, for example using only the previous 2 tags, rather than the whole previous tag sequence. These independence assumptions make estimating the model parameters straightforward to compute using the maximum likelihood estimates based on the frequency counts of seeing the words co-occur with the tags and the tags co-occur with each other.

In this case, we are learning a probability distribution over sequences (x, y), where we assume that we have access to the labels y. However, this same kind of HMM can also be used without the labels to estimate an unsupervised tagging model, a variant of which we will discuss more in Chapter 4.

3.3 Latent Structures

In the POS tagging task, we have looked at an example of supervised learning, where we assume that for each input sequence, we have access to the gold output label sequence to train against. However, later we will consider the case in which the sequence we wish to learn is a latent variable that will be guided toward some different task objective that towards matching a gold label sequence. It is useful then to consider the case where the structured model we wish to learn is represented as a latent variable. In section 3.2 we learned a probability distribution over sequences (x, y) in which we assume that we have access to the labels y. In a latent structured model, we assume that we do not have access to labels for the structure we wish to learn and take a more unsupervised approach, in which we treat the labels as latent states.

A common example of unsupervised learning of latent structures is the task of word alignment in order to generate translation components with which to train machine translation systems. Manually aligned data is expensive to produce, and thus the most common approach to generating word alignments is to infer them in an unsupervised fashion from sentence-align parallel data. Typical approaches to this task use some combination of an unsupervised HMM and the IBM models. In an unsupervised HMM, rather than directly computing the MLE over the observations and labels to train the model parameters, we maximize the likelihood of the observed data marginalizing over all possible settings of the latent variables and optimizing the model using standard methods such as expectation-maximization.

To give a basic introduction to typical approaches to unsupervised latent structure learning for word alignment, we consider IBM Model 1 and how to estimate it using EM. As was briefly noted in 2.2.1, the probability of target sentence $t_1^I = t_1, .., t_I$ being a translation of source sentence $s_1^J = s_1, .., s_J$ is formulated over the latent alignment a between them:

$$\Pr(s_1^I|t_1^J) = \sum_a \Pr(a, s_1^J|t_1^I), \tag{3.6}$$

where a specifies a mapping for each word s_j in s to some target word t_i in t. To estimate the alignment parameters under IBM 1 we start with some initial setting of the model parameters, and then for each sentence in the parallel corpus we consider all possible alignments between the words in that sentence. Given the current parameter settings for $p(t_i|s_j)$, we gather fractional counts for each of these possible alignments and then aggregate them over the number of times we observe each source-target word co-occur in the same parallel sentence in entire corpus. We then update the model parameters towards maximizing the likelihood of the parallel corpus.

This basic alignment model specifies lexical translation probabilities and is frequently used to initialize more complex models that are more expensive to train. Like its more sophisticated variants, as a generative model it jointly models the observed and latent variables with the goal in discovering the optimal setting of the latent variables given the observed data.

3.4 Latent Structured Discriminative Models

We have considered models that use supervised structured learning in section 3.2, as well as unsupervised latent structure learning in section 3.3, both as means of inferring the correct structure for the input observations. We would now like to consider the case in which we want a model that will learn latent structures not as an explicit goal in itself, but rather to inform some other task-based objective. In the models described in 3.2 and 3.3, given the observations x we infer structure y by fitting a joint distribution over (x, y). We will now adjust the notation to fit discriminative models over latent variables. Given observed data X, we will now directly model the probability of the labels Y; we will refer to the latent variables as Z, which we posit to inform the choice of label y for each input x.

Efforts towards using latent structures for discriminative learning pipelined approaches, in which the latent structure is inferred and held fixed while the model parameters are optimized using features from the latent structure. While appealingly simple, this approach is problematic because it allows errors in the structure inference step to be propagated downstream [24]. In addition, we also may wish to consider optimizing our latent structure model for the a specific task. For example, consider the task of how to segment data for machine translation. We may have a data set that does not come pre-annotated with a word-based segmentation (such is often the case with Chinese data), or we may have some parallel corpora in which the two languages are structurally so dissimilar that it may be helpful to have one or both sides broken down into smaller sub-word segments. In this scenario, we may want the data to be segmented in some form, but since the desired goal is to do SMT, we may prefer the segmentation of the data that is most helpful for this task, rather than the gold, linguistically-motivated segmentation. For example, when translating between English and Finnish, we may wish to segment out the Finnish case markers that have explicit lexical equivalents on the English side (such as the genitive case) but not those case markers that have no explicit lexical realization on the English side. Thus, we now consider latent discriminative models that jointly learn the latent parameter settings along with the labels.

[4] present an approach to learning over latent structures to inform some extrinsic task know as learning over constrained latent representations (LCLR). In this framework the latent structures are jointly learned along with a binary discriminative classifier. This model is based on a decision function that defines the score of input x based on some parameter vector $\mathbf{w}$ and searching over Z, the space of possible latent structures for x:

$$f_{\mathbf{w}}(x) = \max_{z \in Z} \mathbf{w} \cdot \phi(x, z) \tag{3.7}$$

To use this as a binary classifier, we assume that examples are positive for which $f_{\mathbf{w}}(x) \geq 0$.

To train a model using this decision function, using some non-decreasing loss function ℓ, we then take as our objective:

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_i \ell(-y_i f_{\mathbf{w}}(x_i)). \tag{3.8}$$

This objective is semi-convex because of the inner maximization over latent structures inside the outer minimization over the weight vector. This model searches for the best possible latent structure while minimizing the overall loss, which makes optimization more complex than the standard convex case. However, it captures the intuition that for negative examples it should be difficult to find a coherent linguistically motivated structure. In order to optimize this objective, we first find the latent structure and then update our model parameters. This is straightforward for the positive examples, since the objective is only nonconvex in the negative case. For the negative examples, we must iteratively search for the latent structure and then update the model weights for each example, using a cutting-plane algorithm that rather than search at each iteration over the space of all possible latent structures instead stores all previously seen latent structures in a cache and searches over that. This method achieves high accuracy on several classification tasks.

[10] presents a method for discriminative optimization for machine translation over latent derivation structures (MIRA), in which the output labels being learned are also structured, in the form of SMT output. This model also tackles a more complex discriminative latent structure task in that the objective is less straightforward that the binary case– instead of directly optimizing against a binary classification loss, we optimize against an automatic translation evaluation metric. In this framework, the objective uses a loss function that combines model loss with the extrinsic loss, as measured by the difference in bleu score between a proposed translation candidate and some reference translation.

To formulate this model, we start with a typical linear max-margin objective over N training examples:

$$L(\mathbf{w}) = \frac{1}{N}\sum_i L_i(\mathbf{w}), \tag{3.9}$$

where

$$L_i(\mathbf{w} = \max_{z \in Z} v_i(\mathbf{w}, z, z_i), \tag{3.10}$$

$$\max_{z \in Z} v_i(\mathbf{w}, z, z_i) = \ell_i(z, z_i) - \mathbf{w} \cdot (\phi(z_i) - \phi(z)), \tag{3.11}$$

$$\ell_i(z, z_i) = B(z, z_i) - B(z', z_i). \tag{3.12}$$

The function B measures the bleu score of a translation candidate versus the reference translation, allowing ℓ to capture the relative bleu loss between any pair of translation candidates.

This objective can be optimized effectively using the MIRA quadratic programming algorithm, which employs a cutting-plane style approach, similar to LCLR. We will discuss this further in 5.2.

Neither LRLC nor MIRA hold fixed latent structures, nor do they make assumptions about what the 'correct' latent structure is, instead they allow the model to be guided in its latent structure learning by what yields the most informative features for the task at hand.

3.5 Summary

In this chapter we have briefly introduced what will be the basic building blocks of our models: structured learning, latent variable modeling, and putting them together in a discriminative framework. In the following chapters we will go on to show how we develop from this framework our models as applied to language modeling over latent shallow syntax and discriminative SMT over joint latent segmentations and alignments.

Chapter 4

Discriminative Language Modeling over Latent Grammatical Structure

In this chapter we introduce latent discriminative learning for language modeling. We begin in section 4.1 by motivating this modeling framework for this task and discussing previous approaches. In section 4.3 we describe our model and the learning algorithms used to train it, as well as the latent structure representations that the model uses. Finall, in seciont 4.4, we report the experiments run along with results and analysis.

4.1 Introduction

Natural language data is implicitly richly structured, and making use of that structure can be valuable in a wide variety of NLP tasks, from machine translation to paraphrase generation to question answering. However, finding these latent structures is a complex task of its own right, enough to warrant extensive study. Early work used a two-phase pipeline process, in which the output of a structure prediction algorithm acts as fixed input features to train a classifier. [5], [18], [24], and [33] have shown that this approach is problematic in that it can propagate error from the fixed latent structures to the classifier, and that it can fail to take full advantage of the labeled data for the final task in generating the latent structures. Moreover, we may wish to allow feedback between the latent learner and the final task learner in optimizing the latent representation of the data for the final task to be performed. More recent work has come to approach latent structure learning for NLP from the perspective of finding the best latent representation for a particular task, rather than what is necessarily the "correct" underlying representation. Work in this vein has focused on jointly learning the latent structures together with the task-specific classifier they inform [9], [4].

[4] in particular set forth a useful framework for solving classification problems using constraints over latent structures, referred to as Learning over Constrained Latent Repre-

sentations (henceforth, LCLR). Inspired by LCLR, we have extended this framework for discriminative joint learning over latent structures to a novel online algorithm. We evaluate the algorithm in comparison to the LCLR batch method on a grammaticality test using a discriminative model that learns over shallow parse (chunk) structures. We show that our online method has standard convergence guarantees for a max-margin learner, but that it attains higher accuracy. Furthermore, in practice we find that it requires fewer passes over the data.

Further, we explore the use of allowing multiple views on the latent structures using different representations in the classifier. This is inspired by [48], who found that using a majority voting approach on multiple representations of the latent structures on a chunking task outperformed both a single representation as well as voting between multiple learning models. We investigate several different methods of combining the views and we show that the multiple-view approach to latent structure learning yields modest improvements over the single-view classifier.

4.2 The Grammaticality Task

To evaluate our algorithms, we use a language modeling task as an example application. Language modeling is an important task in many NLP applications, in which is it usually responsible for making sure that generated text is fluent. The predominant n-gram form of language model (LM) bases the likelihood of generated items upon previously seen word histories. A well-known limitation of n-gram LMs is that they are informed only by the previously seen word-string histories of a fixed maximum length. Thus, while they are sensitive to local disfluencies, they ignore longer distance dependencies between more distant parts of the sentence.

Consider the following example sentences generated by a 3-gram language model:

- chemical waste and pollution control (amendment) bill , all are equal , and , above all else .
- kindergartens are now .

These fragments are composed of viable trigrams, but a human reader could easily judge them to be ungrammatical. However, if our language model were to use latent information such as a syntactic parse of the sentence, our model could recognize their lack of grammaticality based on syntactic cues.

Discriminative models, which have been successfully applied to many natural language tasks, can take into account arbitrary features over a given example, and thus may be able to avoid the shortcomings of n-gram LMs in judging the grammaticality of a piece of text. In the case of language modeling, however, there is no obvious choice of categories between which the model should discriminate. [9] show that by following the pseudo-negative examples approach of [39], they can build a syntactic discriminative LM that learns to distinguish between samples from a corpus generated by human speakers and samples generated by an n-gram model. In this framework, the samples generated from the n-gram model serve as negative examples. While both positive and negative examples will contain probable n-grams, the negative examples can be expected to contain locally fluent constituents but lack global fluency or structural coherence. Thus the discriminative LM learns to assign high scores to the more grammatical structures that look human-generated and low scores to those that do not.

Our approach is similar to [9], but they use probabilistic context-free grammar (PCFG) parses as latent structure, use a latent SVM as the learning model, and handle negative examples differently. We use a latent passive-aggressive (PA) learning algorithm rather than the latent SVM used by [9]. [9] use a PCFG parse as a latent variable for each example. A alternate representation of sentence structure is chunking, which can be seen as a shallow parse. A chunking is represented by tagging each word in the sentence to indicate phrase membership and boundaries, but there are various ways to choose tags to do so. [48] find that voting between several independently trained chunker models using different such tag representations leads to better phrase identification than using only a single representation. Using ideas from [9] and [48], we introduce a discriminative LM which uses one or more chunking representations as latent variables.

We train our model on real sentences versus pseudo-negative sentences sampled from an n-gram model. Our model simultaneously learns to apply multiple sets of chunk tags to produce chunkings representing sentence structure and to prefer the shallow parse features of the human sentences to those sampled from an n-gram LM. Since the shallow parse features of the sampled examples are penalized in the learning algorithm, the chunker-based classifier learns chunking models that prefer not necessarily the linguistically best chunking for each example, but rather the chunking that will assign the widest margin between the positive (grammatical) and negative (ungrammatical) examples.

4.3 Latent Structure Classifier

Our generalized classifier is trained by simultaneously searching for the highest scoring latent structure while classifying data instances. Rather than training the best latent structure to adhere to a linguistically motivated gold-standard, we want to exploit the underlying structure implicit in each example to produce the best classification of the data. Here we use the discriminative latent learning framework due to [4] together with the passive-aggressive (PA) updating strategy due to [17].

4.3.1 PA Learning

The latent structure classifier training uses a decision function that searches for the best structure $z_i^* \in Z(x_i)$ for each training sentence x_i with a space of possible structures $Z(x_i)$ according to feature weights $\mathbf{w}$, i.e.:

$$f_w(x_i) = \arg\max_{z_i} \mathbf{w} \cdot \phi(x_i, z_i) \tag{4.1}$$

where $\phi(x_i, z_i)$ is a feature vector over the sentence-parse pair. The sign of the prediction $y_i^* \mathbf{w} \cdot \phi(x_i, z_i^*)$ determines the classification of the sentence x_i.

Using passive-aggressive max-margin training [17], we incorporate this decision function into our global objective, searching for the $\mathbf{w}$ that minimizes

$$\frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{X} \ell(\mathbf{w} \cdot (f(x_i), y_i)), \tag{4.2}$$

where ℓ is a loss function. Setting ℓ to be hinge loss, at each iteration and for each example x_i we find and update according to a new weight vector $\mathbf{w}'$ that minimizes:

$$\frac{1}{2}\|\mathbf{w} - \mathbf{w}'\|^2 + \tau(1 - y_i(\mathbf{w}' \cdot \phi(x_i, z_i^*)), \tag{4.3}$$

where w is the previous weight vector, z_i^* is the structure found by Eqn. (4.1), $y_i \in \{-1, 1\}$ is the true label (ungrammatical or grammatical) for the example, and $\tau \geq 0$ is a Lagrange multiplier proportional to the example loss. The second term penalizes classification examples proportionally to the degree to which they violate the margin (see Alg. 1).

4.3.2 Optimization Method

Because Eqn. (4.3) contains an inner max over z_i^*, it is not convex for the positive examples, since it is the maximum of a convex function (zero) and a concave function $(1 - y_i(\mathbf{w}' \cdot \phi(x_i, z_i^*))$. In hinge loss, when we drive the inner function to higher values it minimizes the outer problem for negative examples, but maximizes it for the positive ones. So, as in LCLR, we hold the latent structures fixed for the positive examples but can perform inference to solve the inner minimization problem for the negative examples.

4.3.3 Online Training

Our online training method is shown as algorithm 1. It applies the structured prediction and PA update of section 4.3 on a per-example basis in a variant of the cutting plane algorithm discussed in [28]. Since for the positive examples the latent structures are fixed per-iteration, it does a single search and update step for each example at each iteration. For negative examples it repeats the prediction and PA update for each example until the model

```
1  initialize $\mathbf{w}_0$
2  for $t = 0, ..., T-1$ do
3      for each training example $x_i$ in $X$ do
4          repeat
5              find $z_i^* = \arg\max_{z_i} \mathbf{w}_t \cdot \phi(x_i, z_i)$
6              let $y_i^* = \mathbf{w}_t \cdot \phi(x_i, z_i^*)$
7              let loss $l_t = \max\{0, 1 - y_i y_i^*)\}$
8              let multiplier $\tau_t = \frac{l_t}{\|\phi(x_i, z_i^*)\|^2}$
9              update $\mathbf{w}_{t+1} := \mathbf{w}_t + \tau_t y_i \phi(x_i, z_i^*)$
10         until $y_i > 0$ or ($y_i^* = y_i$ if $y_i < 0$);
11 return $\mathbf{w}_T$
```

Algorithm 1: Online PA algorithm for binary classification with latent structures.

correctly predicts the label (i.e. until $y_i^* = y_i$). Since the negative examples should never be associated with good latent structures under the model, we wish to penalize all possible structures for the negative examples. However, because this is intractable to compute, we use the approximation of the single-best structure for each negative example. We re-decode the negative examples until the highest scoring structure is correctly labeled as negative. This approximation is analogous to the handling of inference over negative examples in the batch algorithm described in [4]. In the batch version, however, updates for all negative examples are performed at once and all are re-decoded until no new structures are found for any single negative example. We do not retain any information about negative examples between iterations as do [9] or a cache of negative structures per-example as in LCLR.

Alg. 1 is a soft-margin classifier and we can write down the objective function in terms of a slack variable ξ and assuming hinge loss:

$$\mathbf{w}_{t+1} \quad = \quad \arg\min_{\mathbf{w}} \|\mathbf{w} - \mathbf{w}_t\|^2 + \sum_i \xi_i^2$$

s.t. $\forall_i \max(0, 1 - y_i(\max_{z \in Z(x_i)}(\mathbf{w} \cdot \phi(x_i, z)))) \leq \xi_i$.

The update in line 9 of Alg. 1 is justified using Lagrange multipliers to optimize the objective per iteration as shown in [17]. This is a semi-convex problem which becomes clear if the above objective function is expanded into two special cases: one for positive and one for negative examples. The first max in the above objective means that the objective for positive examples is convex when we fix the value of z while the objective for negative examples is convex even if we search for the maximum z. This is what we do repeatedly for negative examples in Alg. 1 while we make a step in the objective using the $\arg\max$ value of z for all positive examples.

For any weight vector $\mathbf{w}'$, ℓ_t^* is the loss for $\mathbf{w}'$ at round t which we write as

$$\ell_t^* = \max(0, 1 - y_t(\max_{z \in Z(x_t)}(\mathbf{w}' \cdot \phi(x_t, z))))$$

which we compare to the weight vector produced by Alg. 1 in round t which is

$$\ell_t = \max(0, 1 - y_t(\max_{z \in Z(x_t)}(\mathbf{w}_t \cdot \phi(x_t, z))))$$

.

By splitting up the positive and negative examples we adapt the proof from [17] to show that at round t the cumulative squared loss of Alg. 1 on a particular sequence of examples is bounded from above:

$$\sum_{t=1}^{T} \ell_t^2 = \left(\|\mathbf{w}'\| + 2\sqrt{\sum_{t=1}^{T} (\ell_t^*)^2} \right)^2$$

The proof of the above convergence bound on the loss can be derived by combining the split view of positive and negative examples described in [22] and Theorem 3 from [17].

4.3.4 Multiple Views on Latent Representations

To incorporate multiple views on the latent representations, we perform inference separately for each view for each training example.

In our single-view model, the latent structures are provided by a chunker which splits a sentence into phrasal constituents. [48] find that using multiple chunking representations is advantageous for the chunking task itself. Moreover, they demonstrate that the careful selection of latent structure can yield more helpful features for a task-specific classifier. We thus generate separate latent structures for each of their five chunking representations (which are mostly from [45]) at line 5 of Alg. 1.

The addition of multiple views on latent representations retains the same semi-convexity properties of our single-view algorithm. Each view r is a convex function generating some latent structure over training example x_i. Since the combination of the views is just the sum over each function r, the result is another convex function, so the objective function remains the same.

Each of the views use a different representation of the chunk structures, which we will briefly describe here. For more detailed information, please see [45]. Each representation uses a set of tags to label each token in a sentence as belonging to a non-overlapping chunk type. We refer to the chunking schemas as IOB1, IOB2, IOE1, IOE2, and O+C. The total set of available tags for each of the representations are B- (current token begins a chunk), I- (current token is inside a chunk), E- (current token ends a chunk), S- (current token is in a chunk by itself), and O (current token is outside of any chunk). All chunks except O append the part-of-speech tag of the token as a suffix. IOB1, IOB2, IOE1, and IOE2 are variant of inside/outside representations. The IOB representations distinguish only between B, I, and O. IOB1 differs from IOB2 in that it assigns the tag B only if the token that follows is inside a chunk. IOE1 and IOE2 only use E, I, and O, and they differ in that IOE2 assigns

Token	IOB1	IOB2	IOE1	IOE2	O+C
In	O	O	O	O	O
early	I	B	I	I	B
trading	I	I	I	E	E
in	O	O	O	O	O
Hong	I	B	I	I	B
Kong	I	I	E	E	E
Monday	B	B	I	E	S
,	O	O	O	O	O
gold	I	B	I	E	S
was	O	O	O	O	O
quoted	O	O	O	O	O
at	O	O	O	O	O
$	I	B	I	I	B
366.50	I	I	E	E	E
an	B	B	I	I	B
ounce	I	I	I	E	E
.	O	O	O	O	O

Table 4.1: The five different chunking representations for the example sentence "In early trading in Hong Kong Monday , gold was quoted at $ 366.50 an ounce ."

the E tag regardless of whether the token that follows is inside a chunk. Table 4.1 shows the different chunking schemas on an example sentence.

Each of these chunking schemas can be conceived as a different kind of expert. The inside/outside schemas vary over their sensitivity to finding chunks that span multiple tokens. The IOB variants will be better at detecting where a chunk begins, whereas the IOE variants will do better at detecting the chunk's end. O+C allows for a more fine-grained representation of the chunk types.

It is straightforward to use dynamic programming to find the best chunking for each representation. The features of $\phi(x, z)$ are 1-, 2-, 3-grams of words and POS tags paired with the chunk tags thus found, as well as bigrams of chunk tags. Such features are taken across the chunkings for each representation. We use entirely separate chunk tags for each representation. E.g., although each representation uses an "O" tag to indicate a word outside of any phrase, we consider the "O" for each representation to be distinct. Like the single-view version, this model can optionally be initialized by using weights obtained by training for the standard phrase-identification chunking task. The algorithm is given in algorithm 2. In the following sections we describe some different approaches toward using the multiple representations generated by the different views by modifying the combine step on line 7.

```
initialize $\mathbf{w}_0$
for $t = 0, ..., T-1$ do
    for each training example $x_i$ in $X$ do
        repeat
            for view $r \in R$ do
                find $z^*_{i_r} = \arg\max_{z_i} \mathbf{w}_t \cdot \phi(x_i, z_i)$
            let $\phi(x_i, z^*_i) = \text{COMBINE}(\{\phi(x_i, z^*_{i_r}) : r\})$
            let $y^*_i = \mathbf{w}_t \cdot \phi(x_i, z^*_i)$
            let loss $l_t = \max\{0, 1 - y_i y^*_i)\}$
            let multiplier $\tau_t = \frac{l_t}{\|\phi(x_i, z^*_i)\|^2}$
            update $\mathbf{w}_{t+1} := \mathbf{w}_t + \tau_t y_i \phi(x_i, z^*_i)$
        until $y_i > 0$ or ($y^*_i = y_i$ if $y_i < 0$);
return $w$
```
Algorithm 2: Online PA algorithm for binary classification with multiple views latent structures.

```
COMBINE($x_i, \{z^*_{i_r} : r\}$):
for all $r \in R$ do
    convert $z^*_{i_r}$ to common representation $Z_{i_r}$
get consensus structure $\hat{Z}_i$ by voting between $\{Z^*_{i_r} : r\}$
for all $r \in R$ do
    convert $\hat{Z}_i$ into view-specific representation $\hat{z}^*_{i_r}$
return $\phi(x_i, z_i) = \sum_r \phi(x_i, \hat{z}_{i_r})$
```
Algorithm 3: Majority voting method of combining multiple views.

Method 1 - Adding Representations

In our first method, we simply extract the features used in each of the different representations and combine the features for each example x_i. The resulting z^*_i is a tuple of chunking structures, and $\phi(x_i, z_i)$ is a feature vector across all representations, which is then used in the weight vector update. Since the features generated by each representation's inference are distinct to that representation, they are shared in updating the model, but not in inference. Therefore the features contributed by each example are distinct, and $\phi(x_i, z_i)$ can be considered the sum of separate feature vectors for each representation r: $\sum_r \phi(x, z_{i_r})$.

Method 2 - Majority Voting

In the majority voting method, we combine the output of the views by converting the structures found under each representation to a single common representation, from which we then pick the consensus structure and update the model towards that. However, the consensus structure is first converted back to each of the individual representations for the weight vector update step so that we update weights pertaining to features for each of the representational forms.

```
1 COMBINE($x_i, \{z^*_{i_r} : r\}$):
2 for all $r \in R$ do
3     for each representation $s \neq r$ do
4         convert $z_{s_i}$ to common representation $Z_{i_{rs}}$
5     get consensus structure $\hat{Z}_{i_r}$ by voting between all $\{Z_{i_{rs}} : s \neq r\}$
6     convert $\hat{Z}_{i_r}$ into view-specific representation $\hat{z}^*_{i_r}$
7 return $\phi(x_i, z_i) = \sum_r \phi(x_i, \hat{z}_{i_r})$
```

Algorithm 4: Co-training based method of combining multiple views.

Method 3 - Co-training

The final method we examined for combining the multiple views was inspired by co-training [1]. In co-training, the predictions of a weak classifier are bolstered by information supplied by alternative views on the data. Intuitively, we surmise that allowing the alternative views to guide each single view may act as a form of regularization while still retaining the information from the different experts. Our method is similar to co-training in that we allow each view to be informed by the output of the other view. However, it does not obey the strict condition on co-training that the feature sets from each view should be conditionally independent from each other.

In this method, for each training example, we again begin by converting each of the views' best latent structure to a common representation. Then, for each single view $r_i \in r = 1..R$, the best structure for r_i is selected by voting from the other $R-1$ views. That consensus structure is converted into r's representation and added to the feature vector for all views for that example and the weights are updated accordingly.

4.4 Experiments

We trained several settings of our chunker-classifier as well as the batch baseline classifier, as we detail in 4.4.1. We then evaluated the resulting trained models on the grammaticality task.

4.4.1 Training

For the chunkers we used the CONLL 2000 tagset (consisting of 23 constituent chunk tags), modified for the five chunking representations of [48].

For training data we used the English side of the HK Chinese-English parallel corpus. For positive examples, we used 50,000 sentences taken directly from the English side. For negative examples we used the pseudo-negative approach of [39]: we trained a standard 3-gram language model on the 50,000 sentences plus 450,000 additional sentences from the same corpus. From this we sampled 50,000 sentences to create the negative training data set.

This choice of training data is similar to [9]. However, they do not include the positive examples in the sentences used to train the n-gram LM for pseudo-negative generation, and they do not filter the result to use only words from the positive data. We made these choices so as to make the positive and negative training examples more similar in terms of the words used. If the positive and negative examples have substantially different vocabulary, it artifically makes the task much simpler since it makes it possible for the classifier to avoid using the latent features; a basic 1-gram model can distinguish between positives and negatives based on vocabulary alone with high accuracy, but this situation does not approximate the kind of fluency decisions that we are interested in.

The chunker-classifier can either be started with a zero weight vector or with weights from training on the chunking task itself; we tried both initializations. For the latter option we used weights from supervised discriminative training against gold-standard chunking. This supervised training used the same features as the chunker-classifier with the CONLL 2000 chunking tagset, which [48] refer to as the IOB2 representation. To transfer the weights to the classifier, we scaled them to the range of weight values observed after training the zero-initialized chunker-classifier, approximately $[-0.1, 0.1]$ with a single representation and $[-0.01, 0.01]$ with all five representations. This gives non-zero initial weights for some of the features corresponding to the IOB2 representation, but not for any of the other representations.

4.4.2 Batch Baseline

We implemented two batch baselines. The first is a strict implementation of the LCLR algorithm as it appears in [4], with per-outer-iteration example caching (LCLR). The only difference is that we use a Passive-Aggressive large-margin classifier instead of an SVM. In [4], the SVM was trained using a coordinate descent algorithm on the dual [26], while we train our PA algorithm as we specify earlier. This allows us to consistently use the same learning algorithm to compare between batch and online learning with latent variables. However, we found that in practice, this algorithm severely overfits to our application task. The inner loop that repeatedly performs batch updates to the negative examples swayed the model such that while training error approached zero, it disproportionately classified the test data sets as negative. So, we also implemented a variant that skips the inference step in the inner loop. This variant treated the latent structures found in the inference step of the outer loop as fixed, but relabeled and updated accordingly until convergence, where it would resume the next outer iteration. Since this variant was competitive with the online algorithms, we present its results as LCLR-variant in Sections 4.4.3 and 4.4.4. The algorithm is given in Alg. 5.

```
initialize $\mathbf{w}$
for $t = 0, ..., T - 1$ do
    repeat
        for each training example $x_i$ in $X$ do
            find $z_i^* = \arg\max_{z_i} \mathbf{w} \cdot \phi(x_i, z_i)$
        for each training example $x_i$ in $X$ do
            repeat
                let $y_i^* = \mathbf{w} \cdot \phi(x_i, z_i^*)$
                let loss $l = \max\{0, 1 - y_i y_i^*\}$
                let multiplier $\tau = \frac{l}{\|\phi(x_i, z_i^*)\|^2}$
                update $\mathbf{w} := \mathbf{w} + \tau y_i \phi(x_i, z_i^*)$
            until $y_i > 0$ or ($y_i^* = y_i$ if $y_i < 0$);
    until convergence;
return $\mathbf{w}_T$
```

Algorithm 5: Batch-variant PA algorithm for binary classification with latent structures.

Model	Classification Accuracy
LCLR	90.27
LCLR-variant	94.55
online 1-view	98.75
online multi-1	98.70
online multi-2	98.78
online multi-3	98.85

Table 4.2: Classification accuracy (percent) after 40 outer iterations. Multi-1 refers to the additive method of combining representations; Multi-2 refers to majority voting, and Multi-3 refers to co-training.

4.4.3 Evaluation

One way to evaluate discriminative LMs is on the intrinsic classification of distinguishing real grammatical sentences from generated (and presumably ungrammatical) pseudo-negative sentences.

As test data for this task we used the Xinhua data from the English Gigaword corpus. We used the first 3000 sentences as positive examples. For negative examples we trained a 3-gram LM on the first 500,000 examples (including those used for positive data), with no filtering. We used this 3-gram LM to generate five separate 3000 example negative data sets. To account for random variation due to using pseudo-negatives results are reported as a mean over the positive data paired with each negative set.

The results are summarized in Table 4.2.

4.4.4 Analysis

As shown in Table 4.2, all online algorithms outperform the batch versions. We also see a slight increase in classification accuracy with some of the multi-view approaches, in particu-

B	B	(B	I)	B	B
hong	*kong*	*is*	*a*	*conspicuous*	*example*
B	B	B	B	(B	I)

Table 4.3: Output chunking on a sample phrase *"hong kong is a conspicuous example"* from the single-view model (on top) and the co-train based multi-view model (at the bottom). The chunking shown is IOB2 for both models.

lar the co-training based approach. Improved performance of methods 2 and 3 for combining over method 1 might be expected. Since the motivation for using multiple representations is that the model might be guided towards those features that are most helpful to the classification problem, intuitively we can suppose that this is more likely to occur when the views are being driven towards agreement, which is not enforced under method 1. The co-training based method in particular is well-positioned to take advantage of having multiple views on the latent structure. This approach encourages agreement between the views, but by retaining multiple consensus structures, it can capitalize on the different kinds of expertise of the various views, which may be lost in a simple majority voting approach.

The sentence shown in Table 4.3 is taken from the positive examples in the testset; it illustrates the advantage that came from having multiple views on the latent structures. This example was correctly classified by the co-train based model but incorrectly classified by the single-view model. As the latent structure prediction was optimized to distinguish between real and synthetic data (rather than to match a gold-standard chunking), neither chunking reflects the linguistically accurate one. However, we see that there were different groupings between the two models: the single view groups "is a" together in a chunk constituent, whereas the multi-view groups together "conspicuous example". While "is" and "a" may frequently co-occur, "conspicuous" and "example" may together form a more coherent constituent.

Figure 4.1 shows the per-iteration test accuracy of the different models. The batch versions are much slower to improve their testset accuracies. Though LCLR-variant attains more competitive accuracy with the online version than LCLR, it seems to be overfitting past the 35th iteration. The online versions, on the other hand, learn quickly and maintain high accuracies. While all online models eventually reach high accuracies, the voting and co-train multiple-views models reach a high accuracy much faster than the simpler models. In particular, the co-training based method reaches an accuracy over 98% by the fifth iteration; for the single-view and the simple multi-view combine methods, it takes at least twice as many iterations. This indicates that co-training based method using mutiple views on the latent representations gives an advantage when training on a budget. We surmise that this is because the enforced consenses between the multiple views finds latent structures that are helpful to the classification task much more quickly.

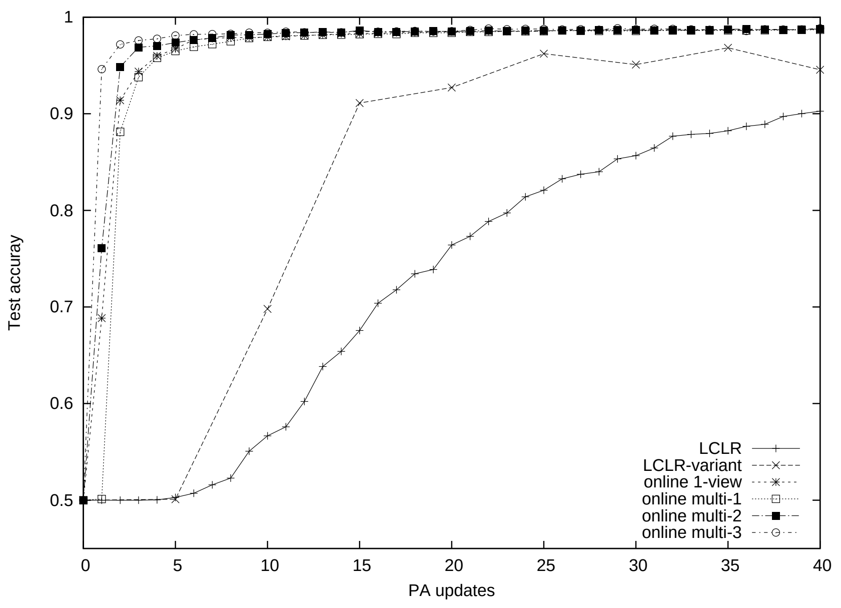


Figure 4.1: Per-iteration test accuracies up to 40 (outer) iterations for the classifiers with weight-vector initialization.

In addition to reaching high accuracy on test data quickly, we also find that in practice, the online algorithm requires fewer updates total in training than the batch version. In the online algorithm, each negative example is re-decoded until it is correctly labeled, whereas in LCLR, all negative examples must be re-decoded each time for any single incorrectly labeled example. We found that for both models, incorrectly labeled examples usually only needed to be re-decoded once to receive a correct label. Because there are generally only a small number of "difficult" examples that need to be re-decoded in each iteration, this means that the batch algorithm is forced to repeat the inference step and perform increased computation on the order of the size of the negative training set.

4.5 Related Work

Our work is most similar to [4], which proposes a batch algorithm for learning over latent representations. We expand upon their framework by developing an efficient online algorithm and exploring learning over multiple views on the latent representations. However, there are several works that have explored a similar task to the one we report on. Max-margin LMs for speech recognition focus on the word prediction task [23, 43, 50]. This focus is also shared by other syntactic LMs [8, 53, 47] which use syntax but rely on supervised data to train their parsers and then decode for LMs from left to right but also bottom-up [6]. [7] and [49] use parsing based LMs for machine translation but LM integration into the decoder limits the number of features used compared to a full-sentence discriminative LM. Our focus is on fully exploiting the latent variables and training whole-sentence discriminative LMs. Our chunker model is related to the semi-Markov model described by [39], but ours can take advantage of latent structures. Our work is related to [9] but differs in ways previously described.

4.6 Conclusion and Future Work

In future work, we plan to apply our algorithms to more tasks to corroborate its effectiveness in other areas. We would also like to undertake more thorough analysis of the properties of online learning algorithms over latent structures. For multiple views in latent structure learning, we have explored separate inference for each representations' view; in the future, we will examine the use of joint inference across multiple latent representations.

Additionally, we intend in future work to conduct an analysis of the convergence bounds of the co-training inspired strategy for combining multiple views on the latent strucures.

We have presented a general online algorithm for learning over constrained latent representations, and we have shown that it attains higher accuracy on a binary grammaticality test. We have also explored the use of multiple views on latent representations via concatenation, majority voting, and co-training. This type of training can be applied to various NLP tasks that stand to benefit from latent structure information.

Chapter 5

Joint Latent Discriminative Segmentation and Alignment for SMT

5.1 Introduction

In this chapter, we discuss our model for joint segmentation and alignment for SMT. We implement and evaluate discriminative max-margin training for machine translation, wherein the alignment and the segmentation of the source data are latent variables. Our model uses fully discriminative training to learn the translation model, whereas much work in discriminative training for MT does only reranking or other adjustment to a generative model. Our model is similar to the discriminative models of [52], [51] and [30], but differs from those works in several ways: first, unlike these works, it requires no precomputed phrase table, but rather learns the phrase correspondences as part of truly end-to-end discriminative training. Second, our work uses a max-margin optimization rather than simple perceptron-like updates, and does so in the full model training, rather than in tuning only. Third, this work models both the alignment as well as the segmentation of the data as latent structures, making it possible to select optimize the segmentation for the translation task.

For a baseline we use the Moses ([25]) phrase-based translation system.

We evaluate our new model and the baselines on a Turkish-English translation task. We consider using the translation system that our model learns as an alignment system that can act as a drop-in replacement for GIZA++ as well as a source of new phrases, features, and weights for the Moses translation system.

5.2 End-to-End Discriminative Model

Here we develop a discriminative translation model that jointly learns the segmentation and translation of the source data, using MIRA-style updates [10]. Our model splits the inference step into two phases: in the first, we optimize over the best segmentation of the source sentence for the translation model given a fixed (source, target) pair from parallel training data. In the second phase, the decoder uses these best segmentations to find the n-best scoring translations. These are then used in discriminative training on a BLEU-based ([41]) loss function, which updates towards some oracle translation. The oracle translation can be selected in a number of ways, most frequently by using either a local or global update strategy, or else by updating towards a 'hope' translation [10] (see section 5.2.4). Thus, the model learns to prefer the segmentations which lead to the highest scoring alignments.

Our model objective seeks to minimize the translation loss over example points $D = \langle x_i, y_i \rangle_1^N$ with respect to BLEU; thus, we search for the parameters $\mathbf{w}$ that minimize:

$$L(\mathbf{w}) = \frac{1}{N} \sum_i L_i(\mathbf{w}),$$

where

$$L_i(\mathbf{w}) = \ell_i(d, d_i) - \mathbf{w} \cdot (\phi(d_i) - \phi(d)),$$

and

$$d(x, y; w) = \arg\max_y \max_{s,h} w \cdot \phi(x, y, s, h)$$

where (x, y) is a parallel sentence pair with x a sequence of source characters and y a sequence of target words, d_i the oracle translation derivation, s is the latent segmentation, h is the latent alignment, w is the weight vector, and $\phi(x, y, s, h)$ is the feature vector for x, y, s, h. ℓ_i is a measure of the extrinsic loss of the candidate against the oracle translation, measured using BLEU scores. This will be discussed further in section 5.2.6.

Our training algorithm is as follows. Given parallel training data $\{(x_i, y_i)\}$, we initialize a weight vector w and an indicator phrase table t. The phrase table t simply determines which phrases are available, while the phrase weights are determined by w. At each iteration we do the following steps for each source–target pair (x_i, y_i):

1. Find the best segmentation. We force decode with fixed x_i and y_i from the parallel training data (where x_i is unsegmented):

$$s_i^* = \arg\max_s \max_h w \cdot \phi(x_i, y_i, s, h)$$

2. Update the phrase table. We take each source-target subsequence alignment feature induced by s_i^* and add it to t.

3. Re-decode the unsegmented x_i (this time without looking at the target reference) using the updated phrase table t, according to the model $f(x; w)$. We take the model's n-best list and compute the BLEU score of each output against the oracle translation d_i. Each y_j in the n-best induces a corresponding assignment of the latent variables: the segmentation s_j and alignment h_j.

4. Update w. Following [10], we optimize the candidate translations in the nbest list, as an approximation for the space of all possible translations, using a cutting plane-style algorithm. To update the weights, minimize

$$\frac{1}{2\eta}|\mathbf{w}' - \mathbf{w}|^2 + \xi_i$$

subject to

$$\ell_i(y, y_i) - \mathbf{w} \cdot (\phi(y_i) - \phi(y)) - \xi_i \leq 0 \quad \forall y \in nbest_i.$$

In this process, segmenting inflected forms may allow the model to find useful translations that it may not otherwise have access to. However, if we have robust statistics on inflected forms, we prefer to keep them unseparated, since the more fine-grained of a segementation we do, the more we expose the model to search errors, especially when word/segment order is different from source to target. We will elaborate further on these steps in the following sections. However, to get a better idea for how this algorithm plays out on the data, let us consider a toy example. Assume we have the segmented parallel Spanish-English sentences given in figure 5.1, where the original forms are shown on the first and third lines and our model receives the (partially) segmented input and output shown on the second and fourth lines.

In the force-decoding step, our model first finds a segmentation of the source and target sides of each parallel sentence that will induce an alignment. During this step, our decoder is not restricted to using a pre-computed phrase table, but rather considers all possible segmentations and alignments of each parallel sentence. In the representation we use, where each token is a morphological segment, this means that our model consider all possible ways of grouping morphemes together, either on the sub-word level or grouping together multiple morphemes into units that can extend beyond the original word boundaries. Some of these segmentation-alignments will be better than others. For example, for sentences (10) and (11), the model may initially find the segmentation-alignments shown in figures 5.2 and 5.3, respectively (tokens that are treated as a single segment are shown grouped in a block).

The alignment features from these segmentations are then added to the phrase table, a few of which are shown in Table 5.1.

(10) voy a traerlo
ir-1SG a traer+ +lo
I will bring it
I will bring it

(11) se aprobó oficialmente
se aprobó oficial+ +mente
it is officially approved
it is official+ +ly approved

(12) vas a traer el regalo
ir-2SG a traer el regalo
you are going to bring the gift
you are going to bring the gift

(13) oficialmente en sesión
oficial+ +mente en sesión
officially in session
official+ +ly in session

Figure 5.1: Example training sentences with some morphological segmentation.

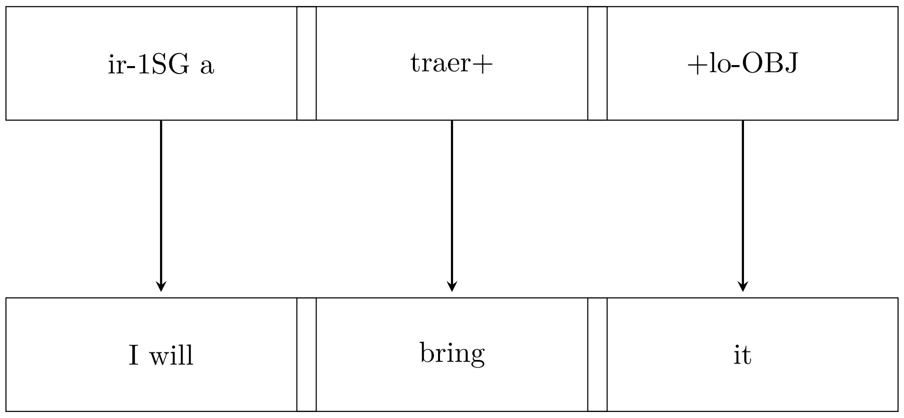


Figure 5.2: A good segmentation produced in the forced decoding step.

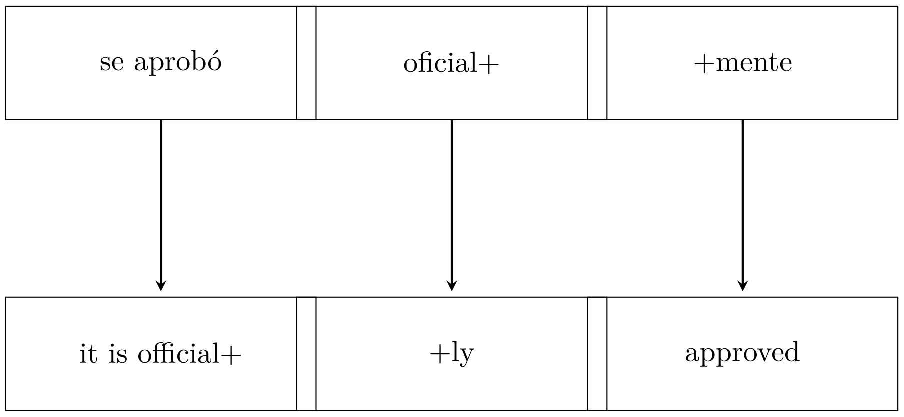


Figure 5.3: A bad segmentation produced in the forced decoding step.

ir-1SG a, I will	0.1
lo-OBJ, it	0.1
mente+, approved	0.1
oficial+, +ly	0.1
se aprobó, it is official+	0.1
traer+, bring	0.1

Table 5.1: Sample phrases in the phrase table extracted from the segmentation-alignment force decoding step with initial weights.

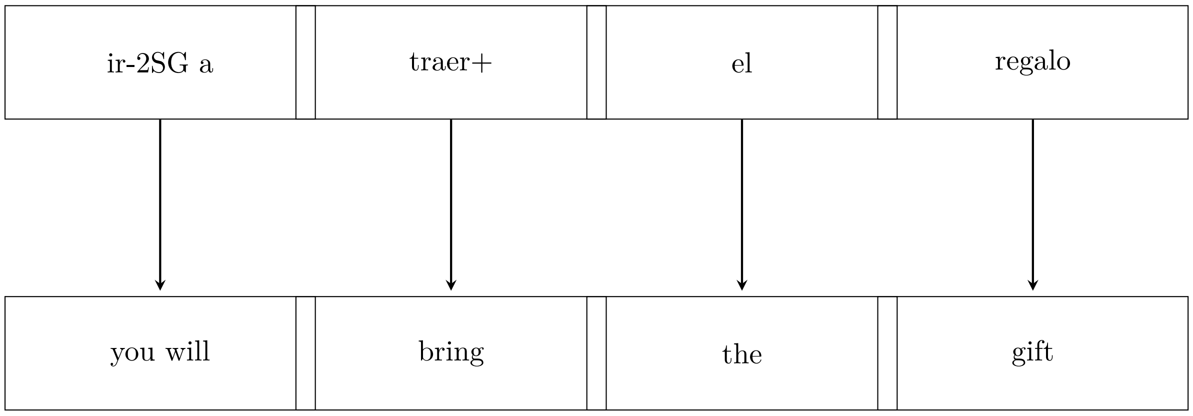


Figure 5.4: A good translation candidate produced in the free decoding step. The features weights for the segmentations used to produced this candidate will be rewarded.

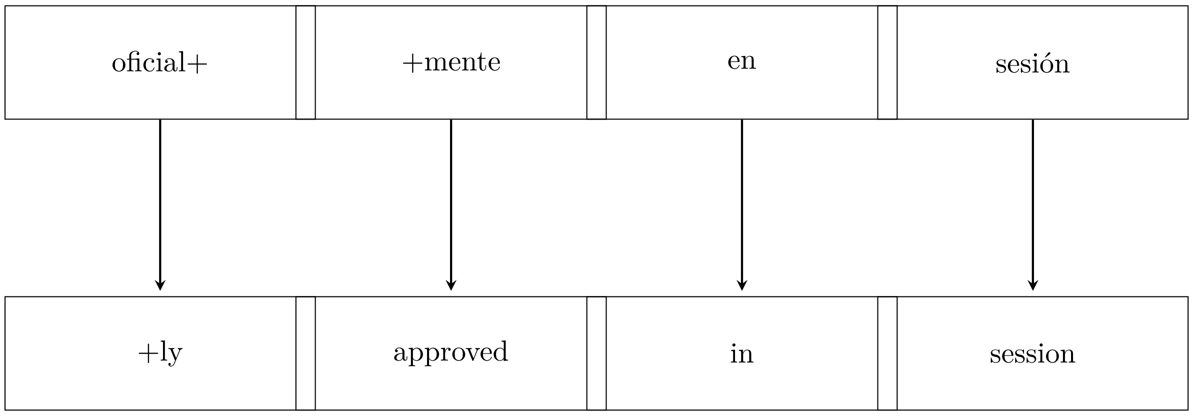


Figure 5.5: A bad translation candidate produced in the free decoding step. The features weights for the segmentations used to produced this candidate will be penalized.

Using this updated phrase table, we re-decode the training data without looking at the target to generate an n-best list of translations. The segmentation features that produce good translation candidates will be upweighted in the learning step and the bad segmentation features will be downweighted. For example, given the phrase table shown in figure 5.1, we might generate the good translation candidate shown in figure 5.4 and the bad translation candidate shown in figure 5.5 for sentences (12) and (13), respectively.

Next, in the weight update step, the features that constitute these translations are extracted, and the features corresponding to the good translation is rewarded, while the features from the bad translation are penalized, as shown in Table 5.2.

In this manner, the model finds the segmentations that lead to helpful alignments for the translation task.

ir-1SG a, I will	0.5
lo-OBJ, it	0.5
mente+, approved	-0.2
oficial+, +ly	-0.2
se aprobó, it is official+	-0.2
traer+, bring	0.5

Table 5.2: Sample phrases added to the phrase table extracted from the segmentation alignment.

5.2.1 Decoding step

The decoding step takes a source sequence s, a phrase table t, and a weight vector w, and produces an n-best list and the features for each translation in the list.

We use a standard stack-based beam decoder with a histogram beam of size l. A hypothesis is defined by the position in the target and the set of covered source sub-sequences. We use a stack for each number of source segments covered and generate the target from left to right, while consuming the source in any order within the distortion limit. To expand a hypothesis, we range over all uncovered source sub-sequences within the maximum source segment length and the distortion limit, and make an expansion for each entry in the phrase table which matches this source sub-sequence. We keep track of all the features which are used by each hypothesis, and score the hypothesis as the sum of the weights of these features in w.

Distortion is defined in the usual way as the absolute difference between the start position of the source sub-sequence being considered and the start position of the last sub-sequence used by the hypothesis being expanded. However, we use a relaxed distortion limit to ensure that there are always at least m uncovered source sub-sequences within the limit: when expanding, we first look for the uncovered source sub-sequences with distortion less than a fixed threshold l. If there are at least m such sub-sequences than we use them. Otherwise we use the m sub-sequences with smallest distortion.

Because early in a run of the algorithm the phrase table is small, it is not always possible for the decoder to find a hypothesis that covers both the source and target completely: it may not have phrases that match at all, or even if there is a valid alignment it may not have any hypothesis within the beam limit that can be completed. This problem is exacerbated by the uniform initial weights, which cause the beam to simply take the first hypothesis expanded. Stacks with several source words covered are likely to end up containing only very similar hypotheses, and if these are unable to be completed, the earlier hypotheses that would be needed to backtrack to the correct alignment may have fallen out of the beam.

To handle these cases we return the n-best list from the last non-empty stack, making early updates [16] from these partial translations. The intuition behind this approach is that the hypotheses that make it impossible to reach a valid translation of the complete source sentence should be penalized by the model. The relaxed distortion limit also helps to avoid some such cases. In later iterations the weights should allow the beam to make better choices.

5.2.2 Segmentation step

The segmentation step takes a source sequence s, a target sequence t (fixed to be the reference translation) no longer than s, and a weight vector w, and produces a k-best list of source-side segmentations with their features.

We use a modification of the beam decoder used for the decoding step. Here hypothesis expansion again ranges over the uncovered source sub-sequences of up to the maximum length, but the target for the phrase is fixed to be the next uncovered word in t. However, we extend our decoder to consider all possible segmentations of the target side as well, as long as the segmentation generates the target in a monotonic order.

In addition, we do not have to limit the alignments to match a phrase table; we instead let it range over all possible segmentations within the segment-length and distortion limits. Some hypotheses can be rejected because they finish covering the target while there is still source remaining uncovered, or because they do not have enough source segments remaining uncovered to align to the remaining target words.

Here, because the source sub-sequences which can be used for expansion are not limited by a phrase table, the only way for the segmenter to be unable to continue is if all hypotheses in some stack are outside the distortion limit, or if all are are rejected as described above. The relaxed distortion limit addresses the former concern, and we have not observed the latter to occur often in practice. None the less, we still allow the segmenter to return partial results if it is unable to complete both sides. We note that the beam search with initial uniform weights will again produce only very similar hypotheses in later stacks, and again rely on the training procedure to correct this by making better weights.

5.2.3 Decoding with the Moses Decoder

Our in-house decoder proved to be too slow to scale to the full datasets. So, we also tried decoding with an out-of-the-box decoder, from the Moses SMT system ([25]). In order to use this decoder with our system, we wrote out all of the phrases our model learns to a Moses-style phrase table, along with the normalized alignment feature weight as the phrase translation probability. We then added a column to the phrase table for each of the additional feature weights for all of the features that were triggered by that phrase pair.

After decoding with the Moses decoder, we extract the features used to generate each example in the nbest list from Moses output. We can then use these feature representations to proceed with learning as usual.

5.2.4 Update strategy

Local updating In the local update strategy, we take the oracle d_i to be the decoder output with the best BLEU score against the reference y_i. In this case, the updates are more conservative, since we encourage the model to prefer translations with higher BLEU scores, but it isn't explicitly aware of the reference translation. However, the danger with this approach is that it requires both a good enough single best translation as well as enough variety in the n-best list to learn to generate outputs that get closer the actual reference. Failing these conditions, the model might not move aggressively enough in the direction of the reference to produce outputs with increasing improvements to BLEU.

Global updating In the global updating strategy, we take the reference translation itself to be the oracle d_i. This updating strategy moves the model more aggressively towards the reference when reachable by the model, though it is often the case that the reference is not reachable. In this case, the model can fail to generate output at all. Therefore, we couple this updating strategy with the early updating approach discussed in 5.2.1. However, [54] argue that this updating strategy is incorrect. Briefly, the intuition behind this is that when we update towards the reference translation itself, rather than some (perhaps partial) candidate produced by our system, it makes it possible for the update to be invalid. To be valid, an update must happen in the cases when a bad candidate gets a higher model score that a good candidate. In global updating, a good partial candidate may be pruned and therefore not reach the final bin, but may rank higher in the model than any (perhaps worse) candidate that survives to the final bin. In this case, while the model prefers the good (pruned) candidate, the optimization step updates against the surviving bad candidate. [54] argue that without ensuring valid updates, the algorithm is not guaranteed to converge.

Updating against the Hope Derivation [10] discusses another updating strategy, referred to as updating towards the 'hope' versus 'fear' candidate translations. The idea behind this updating strategy is that we want to select candidates to update towards and against that take into account a combination of both the model score and the extrinsic loss. So, to summarize, the 'fear' derivation is the one that has the best model score but worst extrinsic score, which the 'hope' derivation is the candidate with the best combined model score and extrinsic score. More formally, these various updating strategies can be contrasted thus, where y is a translation candidate, y_i is the reference translation, $score_{ext}$ is the extrinsic (BLEU) score, and $score_{model}$ is the model score:
Global updating strategy oracle translation:

$$\langle y_i, h^* \rangle = \arg\max h(y_i) \tag{5.1}$$

Local updating strategy oracle translation:

$$\langle y^*, h^* \rangle = \arg\max \langle y, h \rangle score_{ext}(y_i, y) \tag{5.2}$$

Global and local updating strategy update translation:

$$\langle \hat{y}, \hat{h} \rangle = \arg\max \langle y, h \rangle score_{model}(y_i, y) \tag{5.3}$$

Hope and Fear updating strategy oracle (hope) translation:

$$\langle y^+, h^+ \rangle = \arg\max \langle y, h \rangle score_{model}(y_i, y) + score_{ext}(y_i, y) \tag{5.4}$$

Hope and Fear updating strategy update (fear) translation

$$\langle y^-, h^- \rangle = \arg\max \langle y, h \rangle score_{model}(y_i, y) - score_{ext}(y_i, y) \tag{5.5}$$

Using the 'Hope and Fear' updating strategy allows us to maximize the model score, while penalizing the candidates that do this at the expense of the extrinsic score.

5.2.5 Initialization

We consider two ways of initializing the weight vector w. In the first method, we generate features for all possible allowed alignments with all possible allowed segmentations of the source and give them all uniform initial weights. In the second approach, we take segmentations for each training sentence from a separate segmenter, and for each source segment and each target word that appear in the same sentence pair, we give that phrase a uniform weight of one. Given these initial segmentation-based feature weights, we still run the segmenting step before decoding in the first iteration as usual. Thus, while the decoder and segmenter are allowed to generate new alignment features on the fly, the features corresponding to the initial segmentation will be preferred at the outset. The phrase table t is initialized by the segmenting step in the first iteration.

5.2.6 Other Implementation Issues

We found that our implementation of BLEU was a very harsh metric when applied on the sentence level. It was often the case that an initial candidate sentence would contain several correct 1-, 2-, and 3-grams, but no 4-grams from the reference. The negative log weight (approximating log zero) incurred by this would have the effect of cancelling out the scores from the other n-grams, giving too many translations zero BLEU scores to be helpful to the model. Thus, we changed our version of BLEU for use within the training procedure to behave more gently on a sentence-level by doing simple add-one smoothing on the n-gram counts, after the method introduced by [31].

However, this smoothed sentence-level BLEU is still quite a rough approximation of the full BLEU score. Bleu was originally designed to be a corpus-based metric, in which ngram counts are aggregated over the whole corpus. Thus, performing BLEU at the sentence level only may be unnecessarily harsh. So, we followed [52], who approximate corpus-level BLEU using an oracle document. The oracle document is composed of all the previously seen single-best translations for each sentence in the corpus. To measure the score for a translation candidate for a single sentence, we swap the candidate sentence in to the oracle document and take the corpus level BLEU score. The sentence-level score is defined as the change in document-level BLEU from the oracle document to the oracle document with the translation candidate substitution.

5.2.7 Features

We report results for training with different features in the discriminative model. The basic features we consider in all experiments are: source-target aligned phrase pairs, target-side-only monolingual features (to act similarly to a language model in translation), and insertion/deletion features to capture null-aligned tokens. In addition to this, we use features that track of whether a proposed aligned pair/segmentation matches the original word boundaries or whether it instead breaks up a word internally. The intuition with these features is that we want the model to stick to the original word boundaries, except when it has a good reason to break up a word– this is similar to how PBMT memorizes longer translation units/phrases but needs to rely on smaller units when statistics are sparser– so, we initialize these features to reward preserving the original word boundaries and penalize breaking up words.

For example, consider the following parallel English-Spanish sentence:

(14) she likes dogs
le gustan los perros

where our model receives the segmented input:

(15) she like+ +3SG dog+ +PL
le gustar+ +3PL el+ +PL perro+ +PL

Then the word boundary features form a tuple over the word boundary matches on the source and target sides, firing 0 for a match, and 1 for a subword segment. E.g., for the possible phrase pair (‘like+’, ‘gustar+ +3PL’), the word boundary feature would be (1, 0) because it interrupts a word on the source side.

We also use morphosyntactic tag features extracted from the annotations on the Turkish-English corpora provided by a hand-crafted morphological analyzer.

5.3 Experiments

Our model can be used in several different ways in the translation task. We can use it a) as a full end-to-end SMT system, b) to provide alignments for use in a SMT system, for example, as a drop-in replacement for GIZA++, or c) to provide additional phrases, features, and weights to augment the phrase table learned by a conventional SMT system.

For Turkish, we use a data set due to [21] of 50,000 parallel sentences, with 1000-sentence dev and test sets. The data set is annotated with a morphological analysis on both the Turkish and English sides, which provides the morphemes in words with their derivational and inflectional morphology. The English morphological analysis come from the CELEX database; the Turkish morphological analysis comes from [38], a hand-crafted, finite state analyzer designed for Turkish. In the our model's force-decoding segmentation step, we allow it to consider taking morphological segments on their own, or to treat them as part of a larger unit with its surrounding morphological segments. While it is in possible to consider smaller minimal units for the segmentation such as characters, we chose to use the morphological segments as the minimal units instead for two reasons: 1) the model already has a tendency to oversegment the data, so there was no obvious benefit to using smaller units, and 2) character-by-character decoding is excessively slow, because it makes the input sentences on the order of hundreds of tokens in length.

We train each model on the train set from this data and evaluate each by BLEU score on a test set. For evaluation, we use a standard BLEU measure from the multi-bleu script of Moses.

For all trials of the novel model, we used $k = 1$ best segmentations, $n = 500$ best alignments, a distortion limit of $l = 7$ and $m = 10$, a maximum source segment length of $W = 10$, a stack size limit of 100, and an update rate of $C = 1$. For the current experiments we used only global updates. We tested both with and without using the initialization from the external segmentation.

During force-decoding, the decoder searches over all possible segmentations, and when a new segmentation/alignment pair is encountered, this is then added to the model with some corresponding intial weight. However, in successive steps we limit our model to include only the features used in one of the k best segmentations from the force-decoding step, instead of keeping all features explored by the decoder when searching for the segmentation that tiles the parallel sentence. This means that we only have the features that correspond to segmentations that survive the beam and lead to alignments that can cover the sentence pair. This shrinks the model size considerably and speeds up decoding and learning. Using this constraint, the trained model size for Turkish-English was 1,883,125 and 1,211,129 for English-Turkish for the models used in the results given in 5.3.1.

As it is difficult within the scale of this work to build a full end-to-end discriminative SMT system that is competitive with a mature system like Moses, we also consider other ways of using the discriminative model output in the SMT pipeline. For the Turkish-English experiments, we consider using our system's alignment outputs as a replacement for the GIZA++ alignments in Moses. We also consider using the discriminative system's model weights and phrases learned to augment the Moses phrase table.

5.3.1 Augmenting Moses with Discriminative Features

In this set of experiments, we train the discriminative model and then extract the aligned phrases it learns along with their weights and add them to the phrase table learned by Moses. For phrases found in either Moses or the discriminative model but not both, we give them a small non-negative weight of 0.001 to avoid underflow in the Moses translation candidate scoring in MERT.

Discriminative Weights as Dense Features

In introducing the features and their weights that our discriminative model learns into the Moses system, one option is to treat the learned discriminative feature weights like the other dense features that Moses uses, such as the translation probabilities and the language model probability. In this manner, the feature weights our model learns can be added to the Moses translation table for each phrase in the phrase table that also exists in our model, and then the degree to which the dense feature value learned by our discriminative model contributes to Moses' log-linear translation model can be tuned in the standard way, using MERT. We consider three settings of how to use the weights learned by the discriminative model as dense features: 1) taking only the weight learned for the aligned phrase pair; 2) taking the dot product of all the weights with all the discriminative features that fire for an aligned phrase pair; 3) augmenting the Moses phrase table with each of the individual weights separately for all the features that fire for an aligned phrase pair. For the first two of these approaches, this means adding a single extra weight to the Moses phrase table (meaning, 6 phrase weights instead of the typical 5). For the third approach, this means adding as many as 5 extra weights to the Moses phrase table for a total of 10; this is still within the range of the number of feature weights that MERT can stably optimize.

However, out of these three methods for using the discriminative model information in the Moses phrase table, simply using the discriminative feature weight for the aligned phrase performed the best out of the three, so we include results for this method.

Results

Table 5.5 gives the results using the Moses phrase table augmented with the discriminative model phrases and weights for translation from Turkish into English.

Table 5.6 gives the results using the Moses phrase table augmented with the discriminative model phrases and weights for translation from English into Turkish.

system	best BLEU
Moses	26.36
Discrim-Seg+wordboundfeats	**26.85**
Discrim-Seg+wordboundfeats+syntax	26.25

Table 5.3: Turkish-English system BLEU score comparison on test set. Boldface denotes a statistically significant result.

system	best BLEU
Moses	17.90
Discrim-Seg+wordboundfeats	**18.45**
Discrim-Seg+wordboundfeats+syntax	18.33

Table 5.4: English-Turkish system BLEU score comparison on test set

Discriminative Weights as Sparse Features

Another approach towards using the features learned by our discriminative model in Moses is to treat them as sparse features. In this approach, the discriminatively-learned feature values can be individually tuned alongside the dense features using Moses' implementation of batch MIRA training over a development set.

Sparse Feature Results

Table 5.5 gives the results using the Moses phrase table augmented with the discriminative model phrases and weights for translation from Turkish into English using sparse feature tuning.

Table 5.6 gives the results using the Moses phrase table augmented with the discriminative model phrases and weights treated as sparse features for translation from English into Turkish.

5.3.2 Effects of Initialization

We found that the initially segmented model learned faster than the model with uniform segment weights. This is to be expected, as we would predict that the word based seg-

system	best BLEU
Moses	26.36
Discrim-Seg+wordboundfeats	**26.95**

Table 5.5: Turkish-English system BLEU score comparison on test set tuning with sparse features. Boldface denotes a statistically significant result.

system	best BLEU
Moses	17.90
Discrim-Seg+wordboundfeats	**19.00**

Table 5.6: English-Turkish system BLEU score comparison on test set tuning with sparse features.

mentation would contain many segments with direct translation equivalents on the English side, which would be useful to the model. However, we also found that initially rewarding segmentations that match the original word-based segmentation was not enough to get good performance; it was also necessary to add features that capture whether the current segmentation under consideration matches the word-based segmentation in order for the model to learn not to over-segment the data.

5.3.3 Effects of Updating Strategy

The choice of updating strategy was an important factor in model performance (results forthcoming). Under the local updating strategy, the model would stop improving after a few iterations, and would attain much lower BLEU scores than the globally updating model. However, the best performance was achieved using the hope-fear updating strategy. We surmise that it is due to two factors: 1) theoretically, as noted in Section 5.2.4, the global update is not always necessarily valid, since the model's best hypothesis may not survive the beam in search, and 2) practically, the hope-fear updates allow for the most aggressive updates towards the extrinsic loss while still maximizing the model score.

5.3.4 Analysis

There are several patterns that emerge from the experiments using the discriminative model to inform the Moses phrase table. First, we see that the discriminative model consistently helps more with translation into Turkish than into English. This is consistent with our intuitions motivating the use of this model; translation from a morphologically rich language into a morphologically poor language is an inherently information-lossy process. So, we expect that much of the morphological information on the Turkish side can be effectively ignored when translating into a language like English, making the task easier and conventional methods more suitable. However, we see that adding in morphologically motivated segmentations when translating into Turkish yields a more sizeable improvement, ostensibly since retaining access the morphological information *when useful for the translation task* helps produce more complete and fluent Turkish output.

In addition, treating the discriminative model features as dense versus sparse when tuning the Moses system did not make a dramatic difference. We hypothesize that this is due to the twofold nature of the new information that our discrimintative model is contributing the the Moses system. In augmenting Moses with our model features, our model contributes its learned weights over many phrase pairs that already existed in the Moses phrase table. In addition to this, it also contributes new phrases that the Moses system did not learn previous. We conjecture that reason that the sparse versus dense optimization did not matter much was that the contribution of the new phrases learned by the discriminative model outweighed the contribution of the weights it attaches to the phrase pairs.

5.4 Related Work

Much of the previous work in using discriminative models for machine translation has used discriminative training to tune the parameters of a pre-existing generative model [12, 52]. These models are prevalent and perform well, but because they restrict discriminative training to the tuning phase, training of their translation model is not able to incorporate the breadth of features discriminative modeling allows.

[51] and [30] extend the use of discriminative training for MT, in that they train the full translation model discriminatively, but build this on top of pre-extracted parallel phrase alignments constructed using conventional word-alignment and phrase-extraction methods without the use of discriminative modeling. As in this work, all the works above learn the hidden alignments, however they used a fixed segmentation of the data. While their use of word-based phrases allows a degree of flexibility in the segments of the parallel data that are aligned, this flexibility is constrained to the segments induced by conventional word-alignment and phrase-extraction methods.

5.5 Future work

Primary future work to be done for this task is to run experiments using the framework we have developed on other data sets for other languages. We have focused on using this model for translation for morphologically rich languages; however it would be very interesting to apply this model to translation for languages like Chinese in which no word-based segmentation is necessarily present.

Would like to extend our implementation of the discriminative joint segmentation–alignment model to explore a wider range of more expressive features, and thus exploit one of the natural strengths of discriminative models. In particular, we will add in explicit lexicalized reordering features, which can then be easily utilized by the existing Moses decoder. Further, we would like to add in features that capture aspects of the broader context or provenance such as topic features.

We would also like to crossfold validation training, in which we separate data into folds and run just the segmentation and feature extraction step on the first several (training) folds, then run alignment decode step on the held-out (tuning) fold. With this approach we may be able to avoid possible overfitting issues, even on smaller data sets such as the Turkish data. With this we will do a comparison of training and tuning on separate folds versus all on the same data.

In longer term future work, we are interested in exploring how this framework can be used for domain adaptation. Since our framework is designed to learn over latent features of the data, it would be well-suited to learning latent domains and using them for dynamic domain adaptation.

Chapter 6

Latent Topics as Subdomain Indicators

6.1 Introduction

In Chapter 5, we explored discriminative models for joint segmentation and alignment for SMT. One of the strengths of these kinds of models is that they lend themselves to making use arbitrary features that can capture different granularities of information. In this chapter, we specifically delve into methods that leverage document-level information in the SMT task. As a motivating example, consider translating the sentence "He couldn't find a **match**." This sentence provides little guidance on how to translate the word 'match', which could be either a small instrument used to start a fire, or a correspondence between two types of objects. Whether we include word-based, phrasal, or even more long-distance features including syntax or argument structure, the system does not have sufficient information to pick the proper translation. However, if we know that the topic of the document relates to finding medical documents (e.g. transplant donors) rather than starting fires, the system may be able to predict the appropriate translation.

Finer grained document-level information may also be useful in disambiguating the correct translation from multiple candidates in other ways. For example, consider translating the following English-Spanish sentence pair:

(16) the women are not worried
las mujeres no están preocupadas

To translate this English sentence correctly into Spanish, the adjective 'worried' must agree with the gender and number of the subject 'the women.' However, since English does not mark adjectives for gender and number, 'worried' may map to multiple different forms on the Spanish side. Thus from the training data the lexical translation of 'worried' may be ambiguous; we are likely to have multiple possible translations, as shown in Table 6.1.

worried	preocupado
worried	preocupada
worried	preocupados
worried	preocupadas

Table 6.1: Multiple candidate translations from English into Spanish.

However, if we know from the context that this sentence is about some women, and that previously in similar contexts the model prefers the feminine plural form 'preocupadas,' then that may help our model to pick the correctly inflected form, even without access to a syntactic parse or morphological analysis of the text.

Indeed, previous work has shown that both explicit document-level information such as document provenance [11] and implicit domain indicators such as topic models [20] can be helpful for the MT task, when considering both coarse- and fine-grained document-level information.

In this work, we investigate using fine-grained topic information for SMT, focusing on translation into morphologically rich languages. We consider the setting wherein we have a relatively small amount of parallel corpora and therefore only sparse statistics over how to translate from uninflecting English into morphologically rich Turkish. We do this by learning a discriminative topic model that takes into account the bilingual information in the parallel text when estimating topic distributions.

6.2 Latent Topic Models

Following prior work [20], we start with a LDA topic model, in which each document d_i is represented by a mixture of topics z_n, as described in previously in Chapter 3.1. Associated with each topic z_n is a probability distribution generating words $p(w_i|z_n, \beta)$. Given a set of documents, this model learns one topic distribution for each document and a global set of word distribution for each topic to optimize the likelihood of the full dataset.

6.3 Lexical Weighting Models

Lexical weighting models aim to modify the translation probabilities of bilingual aligned word or phrase pairs by a distributions over topics, thus capturing the preferences different topics may have for the translation. This can be computed by conditioning on either the document-level distribution or the document- and token-level posterior distribution. [20] found that the latter results in more peaked document-level topic distributions that were most helpful for MT. Following this approach, the topic-based lexical weights can be estimated using a generative LDA topic model over the source data by incorporating word alignment information according to the following criteria: For aligned word pair (t, s), compute the expected count $t_{z_n}(t, s)$ under topic z_n:

$$t_{z_n}(t, s) = \sum_{d_i \in D} p(z_n|d_i) \sum_{x_j \in d_i} c_j(t, s). \tag{6.1}$$

Then compute the lexical probability conditioned on the topic distribution:

$$p_{z_n}(t, |s) = \frac{t_{z_n}(t, s)}{\sum_t t_{z_n}(t, s)}. \tag{6.2}$$

For the token-topic-conditioning models, we add the additional conditioning context of the source word:

$$t_{z_n}(t, s) = \sum_{d_i \in D} \sum_{x_j \in d_i} \sum_{s_k \sim t \in x_j} p(z_n|d_i, f_k), \tag{6.3}$$

where

$$p(z_n|d_i, s_k) \propto p(s_k|z_n) \cdot p(z_n|d_i). \tag{6.4}$$

These lexical weighting models can then be added as a feature in a log-linear translation model. We compute the lexical weight over all words in a phrase and use it as a feature in phrase-based translation:

$$s_{z_n}(\bar{t}|\bar{s}) = -\log\{p_{z_n}(\bar{t}, |\bar{s}) p(z_n|d)\} \sum_p \lambda_p h_p(\bar{t}, \bar{s}) + \sum_{z_n} \lambda_{z_n} f_{z_n}(\bar{t}|\bar{s}). \tag{6.5}$$

6.3.1 Lexical Weight Feature Functions in SMT

Once we have computed the topic-based lexical weights for the phrase pairs, we consider two different options for using them in the Moses system log-linear translation model. If we have K topics, we use an additional K feature functions in the Moses phrase table. One way to formulate these features is to have one feature corresponding to each topic, i.e., for topics $z_0, .., z_3$ and phrase pair (s, t), $f_0 = p_{z_0}(t|s), .., f_3 = p_{z_3}(t|s)$. In other words, using these features for incorporating the topic lexical weights into Moses gives the informativeness of $z_0, .., z_3$ for all the phrases in the phrase table. This assumes that any given topic will have roughly the same degree of usefulness for all sentences in the dev/test set. However, another option for formulating the topic lexical weight feature functions is to consider them in order of informativeness, that is, in order of how high a probability each topic assigns the the phrase pair in its document context. So, with this method, our feature functions would be $f_0 = \max_{z \in Z} p_z(t|s), .., f_3 = \min_{z \in Z} p_z(t|s)$. In this way, the topic that is dominant in the document gets the most sway in choosing the phrase translation. This approach is better suited to our setting, in which we want to dynamically choose the most informative topic for each new sentence, and so this is the method we use.

6.4 Discriminative Latent Variable Topics

While building generative topic models for lexical weighting for SMT may prove helpful, this approach is not necessarily ideal. When the goal is to use the topic information for the SMT taks, we can safely assume that we have access to some parallel data, since this will be used to train the translation mode. So, ideally our topic model will also take advantage of the target as well as the source side of the parallel text, though this target information is not part of the generative topic model estimation. Efforts to parallel data have resulted in proposed polylingual topic models [35, 42, 27], which seek to model the joint likelihood of the source and target sides of the parallel data.

However, mono- and polylingual generative approaches to topic modeling have some shortcomings for SMT. Recall the formulation of the LDA topic model as a joint distribution over the observed documents $d_{1..D}$ and the latent topics discussed in Chapter 3.1:

$$p(\beta_{1..K}, \theta_{1..D}, z_{1..D}, w_{1..N}) = \prod_K p(\beta_k) \prod_D p(\theta_d) (\prod_N p(z_{d,n}|\theta_{d,k}) p(w_{d,n}|\beta_k, z_{d,n})).. \tag{6.6}$$

Firstly, under this formulation, all the words in a document contribute equally to the estimation of the distribution over latent topics for that document. However, this is not necessarily desirable, as we may suppose that some words will be much more highly indicative of a given topic than others. For example, we may imagine that a noun like ‘thing’ might tell us much less about the topic of a document than a noun like ‘mezzo-soprano,’ and we would prefer a model that gives more weight to more highly indicative terms when estimating the hidden topic assignments.

Secondly, we would like a bilingual topic model to be able to capture the fact that some words are more likely to be translated differently from one topic to another than others. For example, the word ‘thing’ will usually be translated into Spanish as ‘cosa,’ regardless of the topic, whereas the word ‘hot’ must be translated as ‘calór’ in some topics (i.e., the weather) but ‘picante’ in others (i.e., cooking). Under generative topic models, each topic’s distribution over words is learned independently; we would prefer to have shared information across topic distributions when estimating topic assignments in order for the topic model to distinguish between translation pairs that are more topic dependent.

In order to address these issues, we use a discriminative bilingual latent variable topic model that optimizes the conditional likelihood of the target text given the source and the

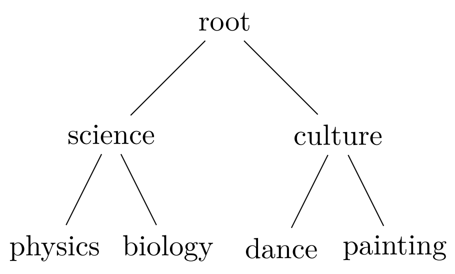


Figure 6.1: A hierarchical organization of topics.

latent topics. This likelihood is formulated as the probability of the latent topic given the whole source document, and the probability of the target given its aligned source and the latent topic, where we use a log-linear model to estimate both of these distributions. More formally, for source document S and target document T with alignment pairs (s, t), we wish to compute the following discriminative model over a mixture of latent topics z:

$$p(T|S) = \sum_z p(z|S) \prod_{(s,t)\in S,T} p(t|s,z). \tag{6.7}$$

The two terms in Equation 6.7 are each formulated as lig-linear distributions, with topic distribution

$$p(z|S) \propto \exp(\theta \cdot f(S,z)), \tag{6.8}$$

and translation distribution

$$p(t|s,z) \propto \exp(\lambda \cdot g(s,z,t)). \tag{6.9}$$

Using these feature-based log-linear distributions, we can allow more strongly discriminative words to have a higher weight in estimating the topic distributions.

In order to capture the idea that some words and alignment pairs are more strongly indicative of particular topics than others, we use a hierarchical topic structure. We formulate the topics in terms of a binary tree structure, where topics go from most general to most specific as we proceed down from the root to the leaf topic nodes. Whenever a feature fires for a topic, all of its more general antecedent topics in the tree also fire. In this way we can learn to set the parameters of more topic-dependent words lower in the tree while the parameters for more topic-agnostic words will only fire higher in the tree for the general topics. For example, if we choose to learn 4 specific topics $1, 2, 3, 4$, then our model will also learn the more general topics $\{1, 2\}$, $\{3, 4\}$, and $\{1, 2, 3, 4\}$. Intuitively, we might suppose that this could correspond to a semantic schema such as that shown in Figure 6.1.

(17) La sopa está caliente
The soup is hot

(18) Hace calór en el verano
It is hot in the summer

Figure 6.2: Example training sentences

	Topic 0	Topic 1
Sentence 1	0.1	0.9
Sentence 2	0.9	0.1

Table 6.2: Per-sentence document topic distributions.

To illustrate what this model learns, consider a small example using the parallel sentences given in Figure 6.2. Using two leaf topics, we would get distributions shown in Table 6.2.

Table 6.3 shows the topic distributions over translations. 'soup' and 'verano' not have topic-dependent translations, but they are strong indicators of topics, enough to sway the per-sentence document topic distribution. This is in turn helpful in influencing the ambiguous translation of 'hot.'

6.4.1 Training

To learn the topic distributions under this formulation, we can use the posterior distribution for a topic z given an aligned source-target document (S, T):

$$p(z|S,T,\theta,\lambda) = \frac{p(z,T|S,\theta,\lambda)}{p(T|S,\theta,\lambda)} \tag{6.10}$$

source	target	Root Topic 01	Topic 0	Topic 1
the	la	1.0	1.0	1.0
hot	caliente	0.4	0.1	0.9
hot	calór	0.6	0.9	0.1
soup	sopa	1.0	1.0	1.0
summer	verano	1.0	1.0	1.0

Table 6.3: Topic distributions over word translations.

To compute the gradient of this posterior, we can use the difference between the expected values of the true counts versus the empirical counts under the current parameter settings, using the partial derivatives of the components of the topic parameters θ and λ respectively:

$$\frac{\partial}{\partial \theta_i} \log P(T|S,\theta,\lambda) = \mathbb{E}_{z \sim |S,T,\theta,\lambda}\, f_i(S,z) - \mathbb{E}_{z'|S,\theta}\, f_i(S,z') \tag{6.11}$$

$$\frac{\partial}{\partial \lambda_i} \log P(T|S,\theta,\lambda) = \mathbb{E}_{z \sim |S,T,\theta,\lambda} \sum_{(s,t)\in(S,T)} \Big(g_i(s,z,t) - \mathbb{E}_{t'|s,z,\lambda}\, g_i(s,z,t') \Big) \tag{6.12}$$

We perform batch gradient descent optimization using RProp [2]. We initialize the model with a vector of random weights and we use L2 regularization. However, we keep the weight on the regularizer small (0.1 maximum) so as to encourage sharply peaked distributions over the topics.

6.4.2 Lexical Weighting with Discriminative Bilingual Topic Models

To compute the lexical weights for the phrase pairs from the topic models, we follow a procedure similar to that described in Section 6.3. To calculate the probability of phrase pair (s,t) from source document S under each leaf topic z_n, we use the following:

$$p_{z_n}(t|s) = p(z_n|S)p(t|s,z_n), \tag{6.13}$$

where

$$p(z_n|S) = \sum_{s \in S} p(z_n|s). \tag{6.14}$$

To get the lexical weights for multiword phrases, we take the product over all the aligned word pairs in the phrase.

6.5 Experiments

We train the discriminative topic model on the same 50,000 sentence English-Turkish parallel corpus used in Chapter 5, this time using the word-based version, rather than the version that has been segmented into morphemes. We also train a Moses phrase-based translation system on this parallel data. We then use two distributions output from the topic model to compute lexical weights for the translation system: 1) a distribution that specifies the topic distribution over each aligned word pair from the aligned corpus, and 2) a distribution that specifies the document-level distribution over topics for each bilingual document in the corpus. We follow [20] and use micro-documents consisting of a single sentence, and we select four as the number of leaf topics on which we train the topic model.

system	best BLEU
baseline	26.36
+lex weights	????

Table 6.4: English-Turkish system BLEU score comparison on test set. Boldface denotes a statistically significant result.

To use the resulting lexical topic weights with Moses, we treat them as additional dense translation weights, with which we augment the pre-trained phrase table. To derive the lexical topic weights for the development and test sets, we compute the distribution over topics for the source side of each per-sentence document in the dev/test set in order to compute Equations 6.14 and 6.13. To incorporate the topic lexical weights as dense features in the Moses phrase table, we use the second method described in Section 6.3.1, wherein $f_0 = \max_{z \in Z} p_z(t|s), .., f_3 = \min_{z \in Z} p_z(t|s)$.

In order to use these dynamically computed topic-based lexical weights in Moses, we have to generate new phrase tables for the dev/test sets that are augmented with the new features. To do this, we first filter the original phrase table for each sentence in the dev/test set. So, since our dev and test sets are each 1,000 sentences long, we get 1,000 new per-sentence phrase tables for each set. Next, we compute the sentence-document-level topic distributions to compute the lexical weights for the phrases in each of the per-sentence-filtered phrase tables. We also annotate all the tokens in each sentence and all the phrases in each sentence-filtered phrase table with a markup of the sentence/document ID. With this markup, we can then concatenate all the per-sentence phrase tables together so as to be able to tune and decode the whole dev/test set at once, while assuring that each sentence-document will only be decoded using the appropriate document-specific lexical weights.

We tuned both the baseline and the system augmented with topic lexical weights using MERT.

6.5.1 Results

Results for the English-Turkish SMT experiments are summarized in Table 6.4. We are currently running experiments using a Moses phrase table that is augmented with lexical weights from a four-topic discrimative hierarchical topic model. Final scores and analysis are forthcoming.

6.6 Conclusion and Future Work

In this chapter, we have demonstrated an approach towards hierarchical, discriminative topic modeling, and we have applied it to an English-Turkish translation task. Since this task requires us to translate from a morphologically poor language into a morphologically rich one, we leverage sentence-level document topic information to help disambiguate the proper target translation among many possible for an uninflecting source word. This model is designed with the translation task in mind, because unlike previous work that either fails to make use of bilingual information, or else jointly models the observed data and the latent topics, this model directly optimizes the conditional likelihood of the target data given the aligned source data and the latent topics.

This use of topic information from sentence-level documents could also be considered a form of dynamic domain adaptation over micro-domains. We would like to extend this work into explicit domain adaptation for SMT, since this approach promises to be particularly helpful in domain adaptation settings where we don't have access ahead of time to knowledge about the domain of our test set. We are also interested in extending this model to a multilingual setting, where by leveraging small multi-parallel corpora and allowing topic information to influence translation preferences, we can boost the alignment quality across languages.

Chapter 7

Training Issues for Large-Scale Structured Models

Chapters 4 and 5 describe scructured discriminative models for NLP tasks and give algorithms for optimizing these models. An important consideration in learning these models is how to perform large-scale training, since inference can be costly and we wish to train our models on large datasets when available. Fortunately, the linear sum objectives used in the models discussed in this thesis make them 'embarassingly parallelizable.' However, the method and effects of sharding the data and running parallel training are not necessarily straightforward. In this chapter we investigate different computational topologies for distributed training for large-scale structured discriminative learning, using a simple perceptron algorithm as our model.

Perceptron training is commonly used to learn structured models for various natural language processing tasks, making it desirable to apply on very large datasets. We implemented a distributed perceptron training algorithm in a MapReduce framework for a structured output chunking task and evaluated it on a cluster and on multi-core systems. Our work extends [34] into two areas of analysis. We compare two methods for implementing the algorithm and find that the choice of topology in the MapReduce operations significantly affects runtime, with the unexpected result that duplicating computation increases overall performance. We also examine the behaviour with different numbers of data shards and find a tradeoff between runtime and accuracy.

While perceptron optimization is simpler than some of the latent structured model training algorithms we have explored in this thesis, it is sufficient to demonstrate general empirial considerations of large-scale distributed structured learning.

7.1 Introduction

The perceptron algorithm [44], in particular the global linear model of [15], has been employed to handle natural language processing tasks such as part-of-speech tagging, parsing, and segmentation. Many such tasks require vast training data to create robust models, making them very time-consuming and computationally intensive to train on. It is natural to look to employing distributed systems to speed up training for these tasks.

We implemented distributed perceptron training following the algorithm of [34] using MapReduce [19], chosen as a simple and common framework for distributed computation. [34] also use MapReduce but do not give details of their implementation or evaluation of choices involved. We achieved similar results to [34] in that our implementation not only is significantly faster than the serial algorithm but also achieves a higher accuracy. [13] also use a parallel setting for training the online learner MIRA [17], but do not discuss the parameter combining topology.

This work has the following novel contributions. We examine two topologies for handling the combination of separately trained weight vectors at each iteration, and find that the choice which duplicates computation leads to a shorter runtime. We also vary the number of data shards and observe a tradeoff between runtime and the final testing accuracy. Unlike previous work, we ran our system on both an HPC cluster and single multi-core systems and find similar results in both environments. We report cache miss statistics for the multi-core setting. These findings promise to help to make effective use of the distributed perceptron for NLP applications.

7.2 Algorithm and MapReduce

Algorithm 6 shows the distributed perceptron of [34]. Here OneEpochPerceptron$(T_i, \mathbf{w})$ is one iteration of standard perceptron training on data T_i with initial weights $\mathbf{w}$, and $\mu_1, ..., \mu_S$ are weight mixing coefficients which in our work we fix to be constant $\mu_i = \frac{1}{S}$ (so that line 5 is just a mean). The data is split into S shards, which can be processed separately. At each iteration the new weight vectors resulting from training on each shard are combined to form a single new weight vector for the next iteration. [34] show that this algorithm has the same convergence guarantees as the serial version, with a worst-case learning rate of a single update per iteration.

MapReduce uses two operations over data which is represented as a distributed collection of key-value pairs. A map operation executes a function on each key-value pair, while a reduce operation executes a function on each set of pairs with the same key. Both operations output a new set of pairs, which is the input to the next operation.

In our implementation we use S processes for computation. The algorithm is initialized with zero weight vectors for each shard as $(i, \mathbf{w})$ pairs with keys $i \in \{1, \ldots, S\}$. The training

```
Require: training data $T = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|T|}$
1: shard $T$ into $S$ pieces $T_1, \ldots, T_S$
2: let $\mathbf{w}$ be the zero weight vector
3: for iteration $n : 1..N$ do
4:     $\mathbf{w}^{(i)} = \text{OneEpochPerceptron}(T_i, \mathbf{w})$
5:     $\mathbf{w} = \sum_i \mu_i \mathbf{w}^{(i)}$
6: end for
7: return $\mathbf{w}$
```

Algorithm 6: Distributed perceptron

step (line 4) is a map operation which for each pair $(i, \mathbf{w})$ performs OneEpochPerceptron$(T_i, \mathbf{w})$ and outputs the new weight vector as a set of feature-weight pairs. Each process handles the training for one shard. The averaging step (part of line 5) is a reduce operation which for feature j averages all S new weights for j and outputs a single feature-weight pair. We let MapReduce choose how to distribute this operation across the processes.

Finally, to start the next iteration it is necessary to combine the feature-weight pairs into $(i, \mathbf{w})$ pairs each with a complete copy of the new weight vector. We call this the combine step, and consider two methods which are described in section 7.6.

We note that although the algorithm as presented uses a single global weight vector, in fact each training map operation needs as input only the weights for features present in its own shard. We did not find a successful way to exploit this, but we do not rule it out for future improvements.

7.3 Experimental setup

We ran our experiments on an HPC cluster and on two multi-core systems (not in the cluster). The cluster consists of 64-bit x86 machines (the precise number of nodes used varied in our experiments), which we could not isolate from the load of other users. The multi-core systems were both AMD Opteron systems with 24 cores over 4 chips. The first was temporarily extracted from the cluster for exclusive testing, and the second was set up for hardware counter measurements and used for detailed bottleneck testing. On the cluster, the processes were distributed across multiple machines; on the two exclusive systems the processes were all on different cores.

Our implementation used the `MapReduce-MPI`[1] library running on top of MPI. We also used the `mpi4py`[2] library for other MPI coordination. Besides the S processes described in section 7.2 our system includes one extra process for logging.

[1]http://www.sandia.gov/~sjplimp/mapreduce.html

[2]http://mpi4py.scipy.org/

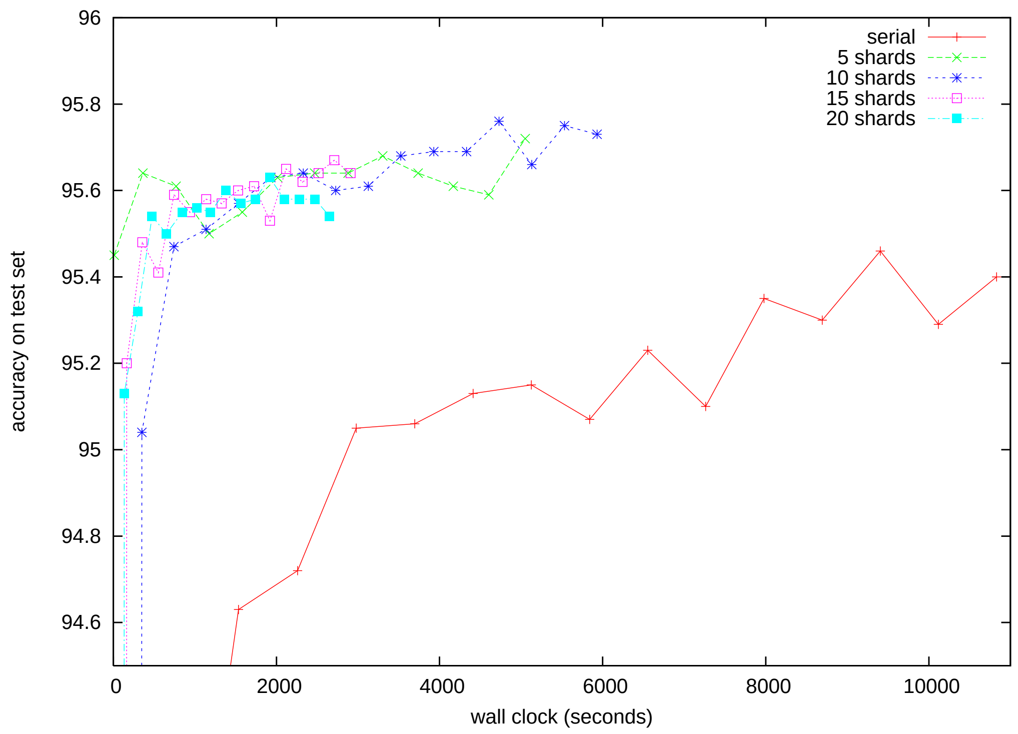


Figure 7.1: Training times for different shardings.

We tested the algorithm on a chunking task using the CONLL-2000 shared task standard data and test set [3]. We present only a single repetition of each experiment here, but we did at least 2-3 repetitions of each experiment and found the results to be consistent on the points presented. In our experiments here we sharded the data by taking equally sized sections in sequential order. The data was made available to all processes on a network file system.

7.4 Convergence and comparison to serial

We first checked convergence by comparing training and testing accuracy for 60 iterations on the cluster (not shown here), with both 5 shards and 20 shards. As expected for a data set which is likely not strictly linearly separable, as the number of training mistakes drops off with increasing iterations the testing accuracy improves up to a point where overfitting and oscillation begin. We observed that 15-30 iterations were sufficient to reach good accuracy, and therefore limited further experiments to this range.

[3] http://www.cnts.ua.ac.be/conll2000/chunking/

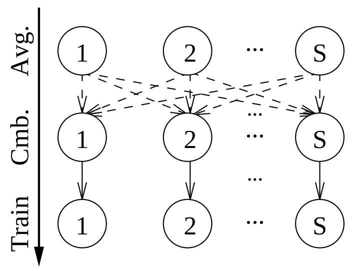


Figure 7.2: Transmission of weights between processes through the averaging, combine, and train steps. Dashed lines represent individual weights and solid lines represent complete weight vectors.

To compare accuracy and runtime, we ran the parallel algorithm with various numbers S of shards and compared against the standard serial algorithm. Figure 7.1 shows results for 15 iterations on the cluster using 2 cores per machine, $\left\lceil \frac{S+1}{2} \right\rceil$ machines, and multi-combine (see section 7.6). Note that while the serial version attains slightly better accuracy than the state-of-the-art of 93.53% [15], the distributed runs are all able to improve on this. [34] also note this behaviour, and suggest that it is due to the averaging effect of combining weight vectors from each shard. We also ran this experiment on the cluster with 1 core and with 4 cores per machine, and on the first exclusive machine, obtaining similar results in each case.

7.5 Number of shards

Although the distributed runs all finish 15 iterations more quickly than the serial version in figure 7.1, in varying the number of shards there appears to be a tradeoff between improvement in runtime and improvement in accuracy; the 10 shard run finds the highest accuracy. It is also interesting to note that while training time decreases as the number of shards increases, there is little difference between 5 and 10 shards, and between 15 and 20 shards, respectively.

We conjecture that the tradeoff may be due to two competing factors. On one hand, averaging across the different weights from training on different shards can act as a regularizer, keeping certain locally useful features from getting too heavily weighted so as to overfit. On the other hand, increasing the number of shards limits the number of features and amount of interaction between features in each shard, making the feature weights less useful.

7.6 Combine step topologies

We consider two methods for the combine step, which we call single-combine and multi-combine. In single-combine, the input feature-weight pairs (the output of the averaging step)

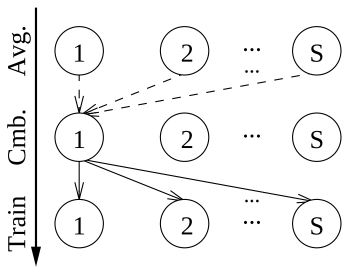


Figure 7.3: Transmission of weights between processes through the averaging, combine, and train steps. Dashed lines represent individual weights and solid lines represent complete weight vectors.

are all sent to a single process, which combines them into a single weight vector and then outputs copies as index-weight vector pairs with indices $1, \ldots, S$. In multi-combine, each input pair is duplicated $S-1$ times so that each process gets a copy and can independently produce a combined weight vector; then each process has a local copy and can begin the next training step without further network activity. We implement both methods by modifying the output of the averaging step and adding a second reduce operation. Figures 7.2 and 7.3 show the communication patterns for the two methods.

Both methods require duplicating and transmitting the same number of weights, although they make the duplication with different granularity and different communication patterns. We initially thought that single-combine might be faster since it avoids duplication of computation, but we found multi-combine to perform faster in practice. Figure 7.4 shows the runtimes for both methods on the cluster, with 2 cores per machine, $\left\lceil \frac{S+1}{2} \right\rceil$ machines, and varying numbers of shards; multi-combine does significantly better in each case.

We believe that this difference is due to single-combine creating a bottleneck at the combining process, which becomes responsible for all the computation and all the transmission of duplicated weights. On the shared cluster we were unable to make bandwidth measurements to evaluate this, so we instead used the second exclusive system. Here the MPI implementation used the multi-core environment directly (not through the network interface). On this system the L3 cache is per-chip while L1 and L2 are per-core. The L3 miss rate is therefore an estimate of traffic between chips, and we used it as an equivalent to network bandwidth for this setting.

Figures 7.5 and 7.6 show miss rates versus time for both methods for 15 iterations and 15 shards. Multi-combine finishes in less than half the time of single-combine. The peaks in both graphs correspond to the combine step, which is the most communication intensive part of the algorithm. Note that multi-combine has a similar miss rate for each process, while in single-combine one process has much higher miss rate than the others. Moreover, the peaks for single-combine are (very roughly) twice as high and twice as long

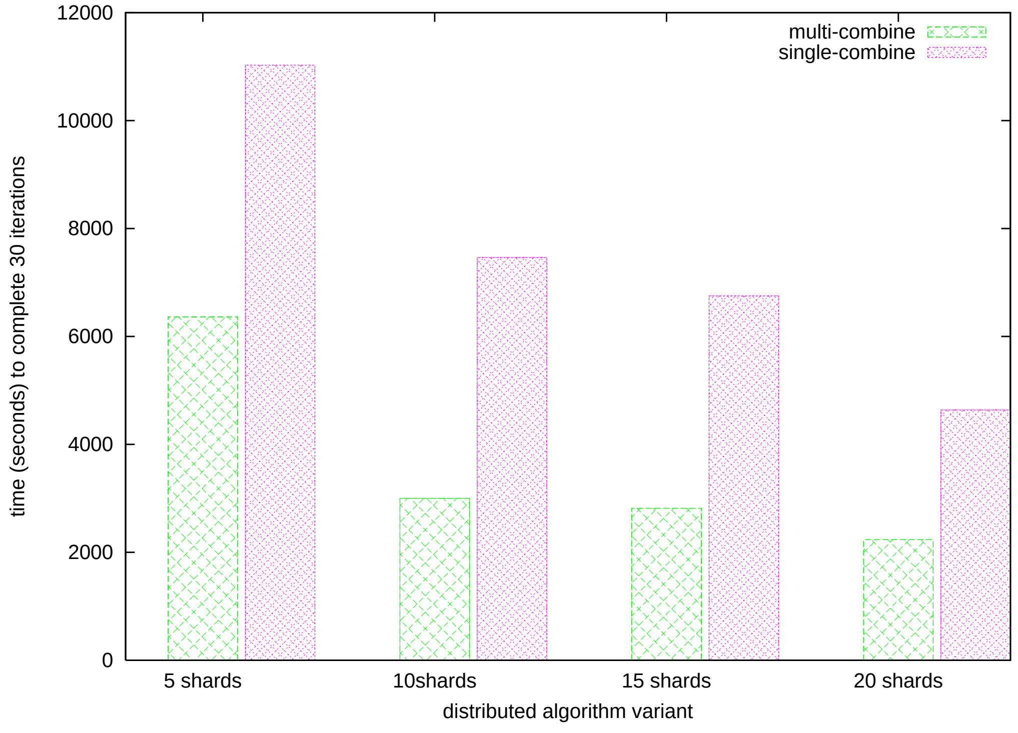


Figure 7.4: Training time for both combining methods.

as those for multi-combine. This suggests that the system is indeed bottlenecked on the single combining process.

7.7 Conclusions

We have shown two useful properties of the implementation of the distributed perceptron for NLP data: duplicating computation when combining weights can improve overall efficiency, while the runtime and accuracy can be balanced by adjusting the number of data shards. We expect that these findings are generalizable for distributed learning for various structured models.

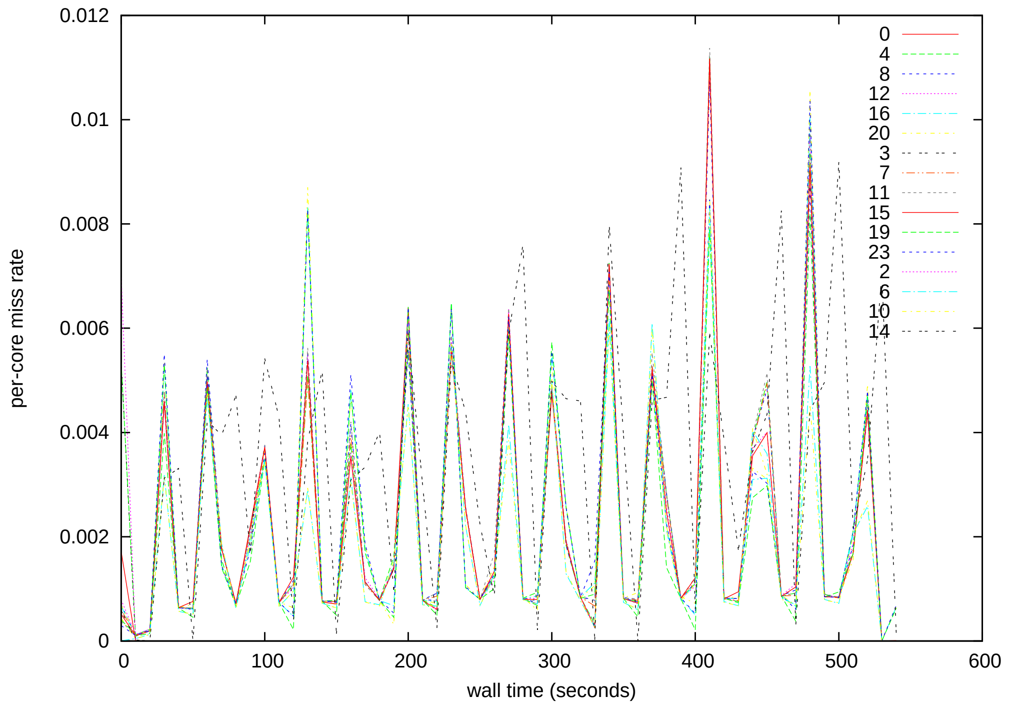


Figure 7.5: Miss rates for the multi-combine method, with a line for each process, labelled by the ID of the core it ran on. (The core IDs and their order correspond to the way in which cores are arranged in the machine.)

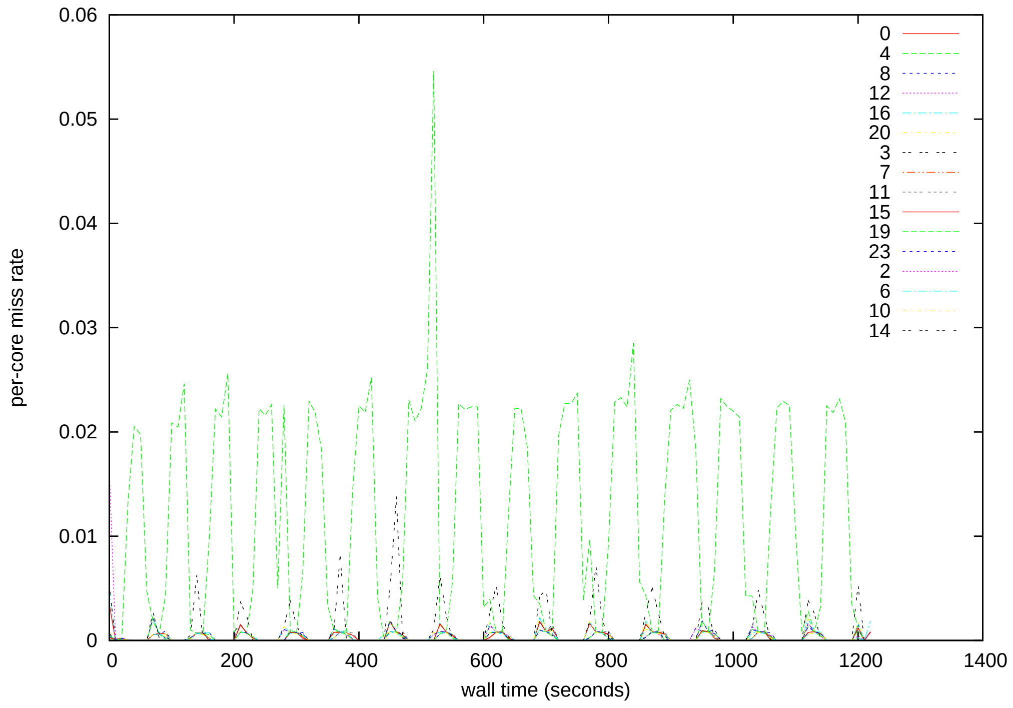


Figure 7.6: Miss rates for the single-combine method, with a line for each process, labelled by the ID of the core it ran on. (The core IDs and their order correspond to the way in which cores are arranged in the machine.)

Chapter 8

Conclusion

8.1 Summary

In this thesis, we have examined using discriminative latent structure learning for NLP. We began in Chapter 2 with an introduction to the tasks we apply this kind of model to in this thesis: language modeling and SMT. We then gave an overview of the learning framework, piece by piece: we separately introduce latent variable models, structured models, and how to put them together in a discriminative framework in Chapter 3. In Chapter 4, we applied our model to a language modeling task framed as a grammaticality classifier. We saw that by training a discriminative model over latent shallow parse structures, we can produce a classifier that can distinguish with a high degree of accuracy between real human-generated text data and pseudonegative data sampled from an n-gram LM. We also looked at multiple different representations of the latent information and examined how to take advantage of using several different latent learners, each with their own representational form. We demonstrated an algorithm that quickly optimizes the semi-convex objective for the binary classifier. In Chapter 5, we considered a more complex objective over structured outputs, where we investigated using this latent discriminative learning framework for SMT. In this task, we allowed the model to range over possible segmentations when searching for how to align sentence tokens, and thus to learn the best segmentation of the parallel data for the SMT task. Training this model can be done effectively using the MIRA algorithm, which is already used to optimize over latent target derivations; we adapted the algorithm to introduce the additional latent segmentation of the source and target alongside the conventional use of the latent derivation of the target. We tested this model in an English-Turkish translation task against a phrase-based SMT baseline, and observed that the addition of the discriminative model features boosted BLEU scores.

We explored how strategies for performing distributed training with structured models, and showed that there are tradeoffs in efficiency and model accuracy in the size and number of shards of data chosen, and that reduplicating the weight averaging computation on multiple nodes can be more efficient than using a single central averaging node to perform and broadcast the averaged model.

For the SMT task, we found that using the joint discriminative segmentation-alignment features in a phrase-based system improves performance, particularly for translation into a morphologically rich language. However, we found that using the discriminative system as a stand-alone SMT system could not achieve results that were competitive with the conventional phrase-based system. We attribute this the the long history of generative aligners such as GIZA++ and the high degree of engineering that has gone into making them achieve high performance for SMT, and we remain hopeful that in the future our system could be more competitive as a stand-alone SMT system with more development.

For the purposes of this work, we have treated the choice of the kind latent structure used to inform the NLP task as fixed. However, this is somewhat arbitrary. Natural language has many different layers of posited latent structure, beyond the kinds we have seen in the work (morphological and shallow grammatical). Full grammatical structure, semantic relations, discursive relations and more can also be recruited to inform NLP tasks. We may also wish to consider latent structure that is less intuitively linguistically interpretable, which is optimized for some particular NLP task without assuming anything about how we model the latent information. This type of approach brings us closer to Neural Network-style approaches towards NLP.

8.2 Contributions

The major contributions of this thesis are in the novel use of latent structures to help with NLP tasks, as demonstrated for language modeling and SMT.

In the language modeling task, we have shown how multiple latent structures can be combined to improve the performance of the grammaticality classifier with fewer iterations of training. While using a single representation of the latent structure already achieves high accuracy, by combining multiple different representations and learners for the latent information, we show that this can be boosted further to achieve high accuracy with very few rounds of training. This process allows the model to effectively select the formalism that most quickly hones in on the most useful distinctions for the grammaticality classifier.

For SMT, we have shown that by jointly optimizing the alignment and the segmentation of the data for the translation task we can attain high quality translations. In particular, in optimizing over the segmentation for SMT, our model novelly considers latent representations of the input, rather than just latent derivations of the output. This approach allows our model to align the representation of the parallel data that produces the best translation output.

8.3 Future Work

For the binary grammaticality classifier in particular, we would like to further extend this work with more exploration of the theoretical properties of the semi-convex training algorithm. In particular, we will conduct an investigation into how the use of multiple latent structures affects the convergence bounds of the algorithm, since the co-training flavored strategy alters the inference step from being a straightforward linear sum over the latent structure output.

For the discriminative joint segmenter-aligner for SMT, we would like to investigate how to add in other sources of latent structural informations. Specifically, we are interested in incorporating the discriminative LM over latent shallow parse structures into the alignment model. We believe this could provide a helpful source of orthogonal information and would be a lightweight way of injecting more useful syntactic information into the model.

In addition, there are several directions in which we would like to extend this general framework in the future that are not task-specific. We have seen in Chapter 4 that learning over the latent structures required performing semi-convex optimization. In future work we would like to explore further analysis of convergence properties with semi-convex learning. On a related note, we are also interested in investigating other approaches to optimization, for example we would like to apply the recent technique of Stochastic Averaged Gradient to explore its properties in the semi-convex case.

We also believe that this type of learning could prove useful in other NLP tasks. We would like to explore applying our methods to, for example, textual entailment and semantic role labeling over latent argument structure.

Bibliography

[1] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, COLT' 98, pages 92–100, New York, NY, USA, 1998. ACM.

[2] Roberto Calandra. Rprop toolbox for MATLAB. `http://www.ias.informatik.tu-darmstadt.de/Research/RpropToolbox`, 2011.

[3] Daniel Cer, Christopher D. Manning, and Daniel Jurafsky. The best lexical metric for phrase-based statistical mt system optimization. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 555–563, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[4] Ming-Wei Chang, Dan Goldwasser, Dan Roth, and Vivek Srikumar. Discriminative learning over constrained latent representations. In *HLT-NAACL*, pages 429–437, 2010.

[5] Ming-Wei Chang, Dan Goldwasser, Dan Roth, and Yuancheng Tu. Unsupervised constraint driven learning for transliteration discovery. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 299–307, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[6] Eugene Charniak. Immediate-head parsing for language models. In *Proc. of ACL 2001*, pages 124–131, Toulouse, France, July 2001. Association for Computational Linguistics.

[7] Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based Language Models for Machine Translation. In *Proc. of MT Summit IX*, 2003.

[8] Ciprian Chelba and Frederick Jelinek. Exploiting syntactic structure for language modeling. In *Proc. of ACL 1998*, pages 225–231, Montreal, Quebec, Canada, August 1998. Association for Computational Linguistics.

[9] Colin Cherry and Chris Quirk. Discriminative, syntactic language modeling through latent SVMs. In *Proc. of AMTA 2008*, 2008.

[10] David Chiang. Hope and fear for discriminative training of statistical translation models. *J. Mach. Learn. Res.*, 13(1):1159–1187, April 2012.

[11] David Chiang, Steve DeNeefe, and Michael Pust. Two easy improvements to lexical weighting. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 455–460, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[12] David Chiang, Kevin Knight, and Wei Wang. 11,001 new features for statistical machine translation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 218–226, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[13] David Chiang, Yuval Marton, and Philip Resnik. Online large-margin training of syntactic and structural translation features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 224–233, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[14] Ann Clifton, Max Whitney, and Anoop Sarkar. An online algorithm for learning over constrained latent representations using multiple views. In *ICJNLP*, Nagoya, Japan, 2013.

[15] Michael Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Empirical Methods in Natural Language Processing - EMNLP*, pages 1–8, 2002.

[16] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[17] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, January 2001.

[18] Dipanjan Das and Noah A. Smith. Paraphrase identification as probabilistic quasi-synchronous recognition. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 468–476, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[19] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[20] Vladimir Eidelman, Jordan Boyd-Graber, and Philip Resnik. Topic models for dynamic translation model adaptation. In *Association for Computational Linguistics*, 2012.

[21] Elif Eyigoz, Daniel Gildea, and Kemal Oflazer. Simultaneous Word-Morpheme Alignment for Statistical Machine Translation. In *Proceedings of NAACL 2013*. ACL, January 2013.

[22] Pedro Felzenszwalb, Ross Girshick, and David McAllester. Cascade object detection with deformable part models. In *Proc. of CVPR 2010*, pages 2241–2248, 2010.

[23] Jianfeng Gao, Hao Yu, Wei Yuan, and Peng Xu. Minimum sample risk methods for language modeling. In *Proc. of ACL*, pages 209–216, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

[24] Dan Goldwasser and Dan Roth. Transliteration as constrained optimization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 353–362, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[25] Hieu Hoang, Alexandra Birch, Chris Callison-burch, Richard Zens, Rwth Aachen, Alexandra Constantin, Marcello Federico, Nicola Bertoldi, Chris Dyer, Brooke Cowan, Wade Shen, Christine Moran, and OndÅŹej Bojar. Moses: Open source toolkit for statistical machine translation. pages 177–180, 2007.

[26] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 408–415. ACM, 2008.

[27] Jagadeesh Jagarlamudi and Hal DaumÃĨ III. Extracting multilingual topics from unaligned comparable corpora. In *Advances in Information Retrieval*, pages 444–456, 2010.

[28] Thorsten Joachims and Chun-Nam John Yu. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76(2-3):179–193, 2009.

[29] Philipp Koehn, Marcello Federico, Wade Shen, Nicola Bertoldi, Ondrej Bojar, Chris Callison-Burch, Brooke Cowan, Chris Dyer, Hieu Hoang, Richard Zens, Alexandra Constantin, Christine Corbett Moran, and Evan Herbst. Open source toolkit for statistical machine translation: factored translation models and confusion network decoding. Technical report, Johns Hopkins University, 2007.

[30] Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation, 2006.

[31] Dekang Lin and Franz Och. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *COLING 2004*, pages 501–507.

[32] Richard Schwartz Linnea Micciulla Matthew Snover, Bonnie Dorr and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*, 2006.

[33] Andrew Mccallum and Kedar Bellare. A conditional random field for discriminatively-trained finite-state string edit distance. In *In Conference on Uncertainty in AI (UAI*, 2005.

[34] Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: North American Meeting of the Association for Computational Linguistics - HLT-NAACL*, pages 1–8, 2002.

[35] David Mimno, Hanna Wallach, Jason Naradowsky, David A. Smith, and Andrew McCallum. Polylingual topic models. In *EMNLP*, 2009.

[36] Sonja Nießen, Franz Josef Och, Gregor Leusch, and Hermann Ney. An evaluation tool for machine translation: Fast evaluation for mt research. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece, May 2000. European Language Resources Association (ELRA). ACL Anthology Identifier: L00-1210.

[37] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[38] Kemal Oflazer. Two-level description of turkish morphology. In *Proceedings of the Sixth Conference on European Chapter of the Association for Computational Linguistics*, EACL '93, pages 472–472, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics.

[39] Daisuke Okanohara and Jun'ichi Tsujii. A discriminative language model with pseudo-negative samples. In *Proc. of ACL 2007*, pages 73–80, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[40] Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *ACL*, 2002.

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

[42] John C. Platt, Kristina Toutanova, and Wen tau Yih. Translingual document representations from discriminative projections. In *EMNLP*, pages 251–261, 2010.

[43] Brian Roark, Murat Saraclar, and Michael Collins. Discriminative n-gram language modeling. *Computer Speech and Language*, 21(2):373–392, 2007.

[44] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, November 1958.

[45] Erik F. Tjong Kim Sang and Jorn Veenstra. Representing text chunks. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, EACL '99, pages 173–179, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics.

[46] Anoop Sarkar and Gholamreza Haffari. Tutorial on inductive semi-supervised learning methods: with applicability to natural language processing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Tutorial Abstracts*, pages 307–308, New York City, USA, June 2006. Association for Computational Linguistics.

[47] Lane Schwartz, Chris Callison-Burch, William Schuler, and Stephen Wu. Incremental syntactic language models for phrase-based translation. In *Proc. of ACL-HLT 2011*, pages 620–631, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[48] Hong Shen and Anoop Sarkar. Voting between multiple data representations for text chunking. In *Canadian Conference on AI*, pages 389–400, 2005.

[49] Libin Shen, Jinxi Xu, and Ralph Weischedel. String-to-dependency statistical machine translation. *Comput. Linguist.*, 36(4):649–671, 2010.

[50] Natasha Singh-Miller and Michael Collins. Trigger-based Language Modeling using a Loss-sensitive Perceptron Algorithm. In *Proc. of ICASSP 2007*, 2007.

[51] Christoph Tillmann and Tong Zhang. A discriminative global training algorithm for statistical mt. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 721–728, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[52] 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.

[53] Peng Xu, Ciprian Chelba, and Frederick Jelinek. A study on richer syntactic dependencies for structured language modeling. In *Proc. of ACL 2002*, pages 191–198, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

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

[55] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.