ABSTRACT

Title of dissertation: LEXICAL FEATURES FOR STATISTICAL MACHINE TRANSLATION

Jacob Devlin, Master of Science, 2009

Dissertation directed by: Professor Bonnie Dorr
Department of Computer Science

In modern phrasal and hierarchical statistical machine translation systems, two major features model translation: *rule translation probabilities* and *lexical smoothing scores*. The rule translation probabilities are computed as maximum likelihood estimates (MLEs) of an entire source (or target) phrase translating to a target (or source) phrase. The lexical smoothing scores are also a likelihood estimate of a source (target) phrase translating to a target (source) phrase, but they are computed using *independent* word-to-word translation probabilities. Intuitively, it would seem that the lexical smoothing score is a less powerful estimate of translation likelihood due to this independence assumption, but I present the somewhat surprising result that lexical smoothing is far more important to the quality of a state-of-the-art hierarchical SMT system than rule translation probabilities. I posit that this is due to a fundamental data sparsity problem: The average word-to-word translation is seen many more times than the average phrase-to-phrase translation, so the word-to-word translation probabilities (or *lexical probabilities*) are far better estimated.

Motivated by this result, I present a number of novel methods for modifying

the lexical probabilities to improve the quality of our MT output. First, I examine two methods of lexical probability *biasing*, where for each test document, a set of secondary lexical probabilities are extracted and interpolated with the primary lexical probability distribution. Biasing each document with the probabilities extracted from its own first-pass decoding output provides a small but consistent gain of about 0.4 BLEU.

Second, I contextualize the lexical probabilities by factoring in additional information such as the previous or next word. The key to the success of this *context-dependent lexical smoothing* is a backoff model, where our "trust" of a context-dependent probability estimation is directly proportional to how many times it was seen in the training. In this way, I avoid the estimation problem seen in translation rules, where the amount of context is high but the probability estimation is inaccurate. When using the surrounding words as context, this feature provides a gain of about 0.6 BLEU on Arabic and Chinese.

Finally, I describe several types of *discriminatively trained* lexical features, along with a new optimization procedure called *Expected-BLEU optimization*. This new optimization procedure is able to robustly estimate weights for thousands of decoding features, which can in effect discriminatively optimize a set of lexical probabilities to maximize BLEU. I also describe two other discriminative feature types, one of which is the part-of-speech analogue to lexical probabilities, and the other of which estimates training corpus weights based on lexical translations. The discriminative features produce a gain of 0.8 BLEU on Arabic and 0.4 BLEU on Chinese.

LEXICAL FEATURES FOR STATISTICAL MACHINE TRANSLATION

by

Jacob Devlin

Thesis submitted to the Faculty of the Graduate School of the University of Maryland, College Park in partial fulfillment of the requirements for the degree of
Master of Science
2009

Advisory Committee:
Professor Bonnie Dorr, Chair/Advisor
Professor Philip Resnik
Professor Clyde Kruskal

Table of Contents

List of Tables

Chapter 1

Introduction

Statistical machine translation began with word-based translation models. The original IBM models [3]—still widely used today for word alignment—estimate the probability of a target word being translated by a source word,[1] as well as word *fertility* (i.e., the number of target words that a source word generates) and *distortion* (i.e., relative movement from one place in the source to another place in the target).

Word-based translation models do not factor in context, so the phrasal model was developed [21]. Using a well-known example, the word "bank" has very different meanings in the sentences "john went to the river bank" and "the bank teller was nice." However, word-based models only use the word-to-word translation probability $P(t|bank)$, which, by definition, does not depend on the context.[2] In an English-to-Spanish system[3], a phrasal system would extract the rules "the bank teller → cajero de banco" and "the river bank → la orilla del rio," implicitly taking this contextualization into account.

[1]The IBM models actually specify a Baysian translation model, and thus estimate target-to-source translation, i.e. $P(f|e)$, where f is a source (or "French") word, and e is a target (or "English") word. However, in modern practice, we find that it is also beneficial to include a probability of the target words given a source word, p(e—f), so I will specify a source-to-target model in all cases to avoid confusion.

[2]This is a simplification—for example, the IBM models use a number of features, such as the aforementioned fertility and distortion—but the point is that they do not factor in the context of "bank."

[3]An MT "system" refers to the entire translation pipeline: training, decoding, optimization, and evaluation. Each of these four processes will be described in later sections.

The phrasal model has limitations relating to reordering and long-distance relationships, so a hierarchical model was developed [7]. For example, given the English-Spanish sentence pair "john, who loved bright colors, drove a yellow car → john, que amaba a los colores brillantes, conducia un coche amarillo", a hierarchical model would extract the rules (1) "john X drove → juan X conducia" and (2) "a X car → un coche X".[4] Rule (1) models the long distance relationship between "juan" and "conducia"—in Spanish, verb conjugation depends on the subject—while rule (2) models the difference in adjective-noun ordering between English and Spanish. Other advanced translation models, such a syntax-based systems [38], have also been created.

This overview of existing translation models has been brief, because the goal of this thesis is *not* to model complex phenomena that a hierarchical system cannot capture. Instead, I look *backwards*, and demonstrate that simple, novel methods for improving the word-to-word translation probabilities (or *lexical probabilities*) can produce significant BLEU[5] [25] gains in a state-of-the-art hierarchical translation system.

The remainder of this chapter provides an overview of the components of the translation system (word alignment, rule extraction, and hierarchical decoding) and details about the training and development data. A roadmap for the subsequent thesis chapters is also given.

[4] "X" represents a non-terminal in a two-sided context-free grammar.
[5] See Section 1.2 and Section 6.1 for a description of BLEU

1.1 Overview of the MT System

The work presented in this thesis was performed as part of the DARPA GALE project, on the BBN team. The primary goal of this project is simply to produce the highest possible quality MT system for two language pairs, Chinese-English and Arabic-English, and four genres: newswire, weblogs, broadcast news, and broadcast conversation. This section presents a brief overview of the thee main processes in BBN's GALE system: word alignment, translation rule extraction, and hierarchical decoding.

1.1.1 Alignment of Parallel Training Data

Given a source/target sentence pair $(\hat{s}_1^J, \hat{t}_1^I)$, an alignment between these two sentences is denoted as a_1^J, where the j^{th} source word is aligned to (i.e. translates to) the ${a_j}^{th}$ target word. The special alignment $a_j = 0$ means that the source word at index j does not align to any target words. GIZA++ [23] with IBM models 1-4 [3] and the HMM model [35] are used to align the parallel training corpus. Under this alignment model, there is a one-to-many mapping between the source words and target words of each training sentence pair. In order to produce a more desirable many-to-many mapping, a "backward" alignment is computed from the target language to the source language, which is done by running GIZA++ again with the source language and target language switched. These alignments are converted into two dimensional matrices and combined using a widely-used join heuristic [20].

The resulting alignment matrix A is a many-to-many mapping between the

source and target words in each sentence pair, where any number of words on either side may be unaligned. A can also be thought of as a $(J+1) \times (I+1)$ binary-valued matrix, where a value of 1 denotes an alignment link between two words. In this case, if a word aligns to no words in the other language, it is said to align to the special word $NULL$. This representation is useful for lexical probability calculation, which is described in Section 3.1. An Example is given in Table 1.1. The aligned parallel corpus, represented as a set of K triples $(\hat{s}_k, \hat{t}_k, A_k)$, is directly used as the input to the rule extraction process.

	NULL	me	gusta	el	coche	mas	rapido
NULL							
i		■					
do	■						
like			■				
the				■			
fast						■	■
car					■		

Table 1.1: Example Spanish-to-English word alignment.

1.1.2 Extraction of Translation Rules

A hierarchical rule extraction process [8] serves as the basis of the next step after alignment. Phrase translations are extracted from each training triple $(\hat{\mathbf{s}}_\mathbf{k}, \hat{\mathbf{t}}_\mathbf{k}, \mathbf{A_k})$. Each possible phrase translation $S \rightarrow T$ is extracted such that no words in S are aligned to any other target words (other than the ones in T), and no words in T are aligned to any other source words (other than the ones in S).

The process for converting the phrase rules to hierarchical rules is to first

convert each phrase pair $S \rightarrow T$ to a double-sided context-free-grammar rule $X : S \rightarrow X : T$[6], and then subtract each phrase pair $S \rightarrow T$ from each rule in the form $X : \gamma_1 S \gamma_2 \rightarrow X : \alpha_1 T \alpha_2$ to create the hierarchical rule $X : \gamma_1 X \gamma_2 \rightarrow X : \alpha_1 X \alpha_2$. In order to reduce the number of possible rules that can be created, restrictions are applied to the size of phrases that can be extracted and subtracted, and filtering is also applied based on the current test set. Joint and marginal counts are then summed over all the rules.

The process for computing the probability of word-to-word translations is much simpler, and merely requires summing over the links in each alignment matrix A_k. These *lexical probabilities* are used in the *lexical smoothing* feature, and will be the focus of a large portion of this thesis. An in depth comparison between the rule translation probabilities and lexical smoothing score is given in Chapter 3.

1.1.3 Decoding

For decoding, a hierarchical model is used that closely follows Chiang's Hiero [8]. In this model, a shared forest of weighted translation rules is created for the sentence being decoded. Since hierarchical rules form a CFG, the test sentence is parsed on the source side using a chart parser similar to CKY thereby creating the shared forest of target derivations. A log linear model is used to score each

[6]The notation here is different from that used in [8] to avoid the confusing use of $\rightarrow$. Here, I write hierarchical rules in the form $X : S \rightarrow Y : T$, where S is the source phrase, T is the target phrase, X is the source non-terminal, and Y is the target non-terminal. I may leave out the left-hand-side non-terminals when they are not necessary for an example. Chiang writes the rules as $X \rightarrow \langle S, T \rangle$, where $\rightarrow$ is used in the normal CFG sense. However, in that notation there is only a single non-terminal, whereas in the current system the source and target can (and usually do) have separate non-terminal ids.

translation rule:

$$\begin{aligned} w(X : S \rightarrow X : T) &= \prod_i \phi_i(X : S \rightarrow X : T)^{\delta_i} \\ \log(w(X : S \rightarrow X : T)) &= \sum_i \delta_i \phi_i(X : S \rightarrow X : T) \end{aligned}$$

Where δ_i is the i^{th} *feature weight*, and $\phi_i(R)$ is the i^{th} *feature score* of rule R. The score of a *hypothesis* (i.e. a full sentence translation) is thus the sum of the log-scores of each rule used in that hypothesis. Each rule can have an arbitrary number of features, and the feature weights (δ_i) are generally determined automatically through optimization towards an evaluation metric such as BLEU [25]. Basic features (ϕ_i) include:

- Language model score
- $P(S|T)$ [13] (i.e., forward rule translation probability)
- $P(T|S)$ (i.e., backwards rule translation probability)
- Lexical smoothing
- Word penalty, i.e. a feature equal to the number of words in α

The hypothesis with the highest log-linear decoding score is chosen by the system as the *MT output* (also known as the *1-best hypothesis*).

1.2 Evaluation of Machine Translation

In order to judge how useful a particular change to the MT system is, it is necessary to automatically and quantitatively evaluate the quality of the MT output using an *evaluation metric*. These evaluation metrics work by comparing each output sentence to its corresponding *reference translation*—a human-produced translation of a test sentence—and producing a normalized score that represents the translation quality of an entire test set. In this thesis, I present results on the BLEU evaluation metric, which counts the number of words and phrases in the MT output sentence that also appear in the reference translation. The resulting BLEU score is a number between 0 and 100, the higher the better.[7]

There is no universal standard for what is considered a significant[8] gain in BLEU, but generally +0.5 BLEU is considered the lower bound for real-world significance. Note that due to the nature of MT, results can only be compared against a team's *own* baseline.[9] This is different from other fields (e.g., facial recognition), where the results on a standard test set are directly comparable across systems.

As mentioned in the last section, the decoding weights are choosen by optimizing them to maximize an evaluation metric score on some tuning set. It is standard in MT literature to present the metric used for optimization as the "primary" evaluation metric. In other words, the system used in this thesis optimizes towards

[7] To give a more formal description, the BLEU score is the geometric mean of the 1-gram to 4-gram overlap percentage between the output and reference (where the denominator is the number of *MT output* n-grams, not *reference* n-grams), multiplied by a brevity penalty to prevent short output. See Section 6.1 for a formal presentation of the BLEU formula.

[8] Not "significant" in the formal statistical sense, but more in the "publishable" sense.

[9] Which is just defined to be the system *without* whatever change is being presented.

BLEU, so the BLEU scores are thus the primary scores examined when deciding whether a particular feature is useful or not. Some papers also report results on alternate metrics such as TER [29] or METEOR [1], but that is not done here for readability sake, due to the large number of conditions and test sets presented.

1.3 Description of Data

This section describes both the training data and development data.

1.3.1 Training Data

The parallel training data used for the experiments reported in this thesis consist of about 200 million (English) tokens in both Arabic-English and Chinese-English. For Arabic, the "Sakhr" data set contains Arabic sentences which have been automatically translated into English by an MT system, and then manually corrected to be accurate and fluent. The composition of the Arabic-English and Chinese-English data is shown in Table 1.2 and Table 1.3.

Data Origin	**Style**	**Size (K tokens)**
LDC	U. Nations	118049
	Newswire	46213
	Treebank	977
	Web	478
	Broad. News	573
	Broad. Conv.	1003
Web-found text	Lexicons	436
	Quran	406
Sakhr	Newswire	30790
Total		**198925**

Table 1.2: Composition of the Arabic-English parallel corpus used for MT training.

Data Origin	Style	Size (K tokens)
LDC	U. Nations	107210
	Newswire	90248
	Treebank	112
	Web	612
	Broad. News	1827
	Broad. Conv.	1346
	Lexicons	222
Web-found text	Lexicons	190
Total		**201771**

Table 1.3: Composition of the Chinese-English parallel corpus used for MT training.

A 5-gram language model is trained on several billions words of English, primarily from the LDC GigaWord corpus and GoogleNews.

1.3.2 Development Data

Our Chinese-English and Arabic-English translation systems are run on four different classes of data: newswire, web, broadcast news, and broadcast conversation. However, development is primarily done on newswire, so only newswire results will be presented in this thesis. For features with positive results, results are presented on both Chinese-English and Arabic-English. For features that produced negative results or were otherwise impractical, I only present results for one language or the other.

For each language, results are presented on one tuning set and two validation sets. The sets are created as a mixture of NIST MT04, MT05, MT06, and MT08 evaluation sets, the GALE Phase 1 (P1) and Phase 2 (P2) evaluation sets, and the GALE P2, P3 and P4 development sets. The size and reference information for each

set is given in Table 1.4. In Section 6.2, I use a slightly different Chinese dev set (which has the GALE P4 data added), but the composition is comparable.

	Chinese Newswire			**Arabic Newswire**		
	#segs	#tokens	#refs/seg	#segs	#tokens	#refs/seg
Tune	1994	76292	4	2114	56599	4
Test 1	1031	40789	1	1171	32330	1
Test 2	2118	79927	4	2093	55203	4

Table 1.4: Characteristics of the tuning (Tune) and validation (Test 1 and Test 2) sets used for development on Arabic Newswire and Chinese Newswire.

1.4 Thesis Organization

In Chapter 2, I describe work related to this thesis.

In Chapter 3, I present an in-depth comparison between rule translation probabilities (e.g., P(cajero de banco|the bank teller)) and lexical translation probabilities (e.g., P(banco|bank)). I first explain how lexical probabilities are computed and used in decoding. Instead of being used as direct features like the rule probabilities[10], they are used in the so-called *lexical smoothing* formula, a simple algebraic function computed on each translation rule.

Next, the relative "importance" of each feature is examined—that is to say, how much it damages the system to remove rule probabilities and/or lexical smoothing. Here, I present the somewhat counterintuitive result that lexical smoothing is significantly more important than the rule probabilities, in spite of the fact that the rule probabilities implicitly model so much more information than the lexical probabilities. Specifically, turning off the rule probabilities (and re-tuning the sys-

[10]In this thesis, "rule translation probabilities" and "rule probabilities" are synonymous, as are "lexical translation probabilities" and "lexical probabilities."

tem) degrades results by 1.5 BLEU, while turning off the lexical smoothing degrades results by 3.5 BLEU. I postulate that this is because the rule probabilities are, in general, poorly estimated, while the lexical probabilities are generally well estimated. Empirical evidence is provided to support this hypothesis.

In Chapter 4, two methods of *biasing* the lexical probabilities are presented. In a general sense, *biasing* the lexical probabilities means to interpolate the lexical probability distribution extracted from the training, denoted as T, with some other lexical probability distribution denoted as B. This is different from simply adding B to the training because B can be different for each test sentence (or test document).

For the first method, *lexical semantic biasing*, each test document is biased with its n most similar parallel training documents. These data are already included in T, but through biasing they are weighted significantly more than would normally be the case. The similarity measure between each test document and bi-lingual training document is computed using a high-performing cross-lingual information retrieval (CLIR) system [37], although *cross-lingual* IR is not strictly necessary as the training documents are bi-lingual. For the second method, *lexical self-adaptation*, each document is biased with its own output from a first-pass decode. This self-adaptation procedure produces a small but consistent gain. A thorough analysis is provided regarding what is causing the gain for self-adaptation, and this is used as a springboard for the next chapter.

Chapter 5 describes an explicit generative model for improving the lexical probabilities, called *context-dependent lexical smoothing*. Here, the accuracy of the lexical translations is improved by using local context—for example, the "previous

word" context would model the probability $P(t|s_i, s_{i-1})$. The key to this feature is the simple but effective *backoff model*—we trust each context-dependent probability $P(t|s + context)$ proportionally to how many times we've seen $s + context$ in the training. In order to mathematically represent this trust model, a *count-based interpolation* is used; this approach has a number of beneficial properties that standard probability interpolation does not.

Chapter 6 describes *discriminatively trained* lexical features. With discriminative training, probabilities are not estimated with a generative story (e.g., the lexical probabilities described above), but instead are estimated as decoding features, meaning their values are chosen to directly maximize the BLEU score on a tuning set.

Chapter 2

Related Work

My analysis of lexical smoothing is somewhat reminiscent of a well-known paper [24], which implemented a large number of complex features using parsers, treebanks, re-ordering templates, and HMMs. The only feature to show a non-trivial gain was the use of IBM Model 1—the oldest and simplest statistical translation model.

There has been some prior work in smoothing the rule probability estimates, which I assert in Section 3.2 would be a difficult task. Foster et al. [12] presents a number of methods for smoothing and backing-off the phrase table estimates, such as Good-Turing discounting and lower-order phrase count estimation through wildcards. Although they report significant gains, closer inspection reveals that the highest-performing method is exactly equivalent to the long-standing context-free lexical smoothing formula used by our system (and many others), and no other methods get a gain on top of this. In other words, none of the smoothing methods worked—they were not able to do better than using unsmoothed rule probabilities and the standard lexical smoothing feature.[1]

Work similar to my lexical probability biasing has been performed on the

[1]This is in reference to the *large scale* Chinese-English experiment of Foster et al. [12]. They also performed several experiments on three "relatively small corpora" where they *did* see a gain using the more advanced smoothing methods, but those results are not relevant to our experimental conditions.

translation rules and language model. Snover et al. [30] demonstrated success with semantic biasing of the language model and translation rules on the document level (i.e., the biasing corpus B is different for each test document). In that case, however, B was selected from *monolingual* data in the target language. The monolingual corpus—used to train the 5-gram language model—is many times larger and has data from far more sources than the parallel corpus, so it is much more likely that the data in the biasing documents will be closely related to the test document. The problem with using monolingual data is that there is no source translation, so lexical probabilities cannot be extracted.[2]

I only report results on lexical self-adaptation in this thesis, but I had previously attempted to perform self-adaptation on the translation rules and language model without success. However, others have reported positive results on rule-level self-adaptation [34, 6]. Ueffing [34] suggests that the BLEU gains from rule-level SA are explained by the creation of *new* rules never seen in the training. For lexical probability self-adaptation, the creation of new lexical translations is not possible, so this cannot be the underlying cause of the gain.

Several papers have described methods for improving MT results through the use of local context [31, 15, 14], but the focus—and therefore methodology—of these papers is somewhat different from that described in Chapter 5. In my CLS model,

[2]For those curious as to how Snover et al. [30] performed rule biasing with monolingual data: A "biased rule" was created between *every* m word phrase in the test document and n word phrase in the biasing document. In an essential step, the lexical smoothing score was calculated on each of these rules using the lexical probability distribution from the training, and this lexical smoothing score was used as a decoding feature. It would clearly be problematic to perform this process for the lexical probabilities, since it assumes the lexical probabilities are already well estimated between the source and target phrases.

the *accuracy* of the probability estimates is the major concern, even if ensuring accuracy means using less context. In the paper most similar to this thesis, Gimpel and Smith [14] add local context to the *rules*, and interpolate the context-dependent rule probabilities using fixed interpolation weights.[3]

The first problem with the method of Gimpel and Smith [14] is that if the rule probabilities are poorly estimated to begin with, then certainly the rule probabilities *plus context* are going to be even more poorly estimated. The second problem is that by using fixed-weight probability interpolation, they capture no information about the reliability of the estimate (e.g., $\frac{1}{2}$ has the same effect as $\frac{500}{1000}$). The results of [14] actually support the ideas presented in this thesis: they were able get a very large 3.0 BLEU gain on UN test data, because they had a very large amount of UN training, but got *no* gain on newswire, because they had much less newswire data. The authors explain their negative newswire results by saying "unsurprisingly, the context features were too sparse to be helpful." This thesis addresses these problems by adding context to the lexical probabilities instead of the rules, and using a count-based backoff model.

Several papers based on work from the Word Sense Disambiguation (WSD) literature [5, 4] are quite similar in spirit to my context-dependent lexical smoothing. A standard WSD system performs supervised learning on sense-labeled training data, and then attempts to predict the senses of words in an input sentence given predefined sense categories. For example, given the input sentence "I deposited money

[3]They use each context-dependent rule probability as a different decoding feature, so the weights are *globally* optimizable, but fixed for the purposes of decoding.

at the bank" and the categories "Senses(bank) = {MONETARY_INSTITUTION, RIVER_BANK, TO_RELY}," what is the sense of "bank"? A WSD system can be used *without modification* to perform context-dependent lexical translation prediction by using the parallel corpus as labeled training data, where the target translations are the senses. For example, "Senses(coche) = {CAR, AUTOMOBILE, BUS}." The WSD system would then be used to predict $P(\text{CAR}|\text{ coche, Context(coche)})$.

The issue is that WSD is a *classification* task, where the goal is to pick the *best* possible sense for a given word in a given context. This is not particularly useful in an MT system, which instead requires a *probability distribution* over all possible senses. Although some classifiers, such as Naive Bayes, naturally produce a true probability distributions, others, such as SVMs and decision tress, produce no numerical scores at all. The classifiers used by Carpuat and Wu [5] do produce numerical scores, but the authors seem to assume that using these raw scores (or even normalizing them to create posteriors) will produce a robust probability distribution, just because the classifier is robust—this is not a valid assumption. In this thesis, the CLS scores are computing using true probabilities, so this not a concern. Empirically, both Carpuat and Wu [5] and Cabezas and Resnik [4] get either negative results or very small gains when performing WSD on single source words.[4]

My work on discriminative features was originally inspired by Chiang et al. [9], although both the optimization metric and features used here are significantly

[4]Carpuat and Wu [5] does get positive results by effectively concatenating *multi-word phrases* together to create a "word" in each language, and running WSD on that.

different from the ones used in that paper. The similarity is that in both papers there is nothing *pragmatically* discriminative about either the features or optimization procedure. In other words, the procedure is discriminative only because of *what* is being estimated, not *how* it is being estimated—I'm simply adding more features to a log-linear decoding model. The only reason a new optimizer is necessary is that our existing optimizer becomes unreliable*In this context, "unreliable" means that the weights choosen by the optimizer will often produce poor results when used to re-decode.* when a large number of features are used—not because it is fundamentally unsuitable for this type of "discriminative" training. Other recent papers have taken a more classically discriminative approach to MT training [32, 18, 2], but most of these procedures would not easily fit with our system.

As was just mentioned, the major impediment to our use of discriminative features in decoding was our long-standing optimization procedure, which was based on Powell's method [28]. We have empirically found that 1-best BLEU optimization with Powell's method becomes completely unreliable when a large number[5] features are used, which makes using a large number of discriminative features out of the question. To address this problem, as we will see in Section 6.1, I implemented a new optimization procedure called *Expected-BLEU optimization*, which is fast and robust for tens of thousands of features.

This procedure is similar to the lattice-based continuous BLEU optimization described in Pauls et al. [26], however it was developed independently, and most

[5]Empirically, we have found Powell's method to have significant problems optimizing just $\sim$25 features.

work was completed before that paper was published. An earlier paper, Tromble et al. [33] also estimates expected BLEU (on a lattice like [26], rather than an n-best list like this thesis), but the authors use a linear approximation of BLEU, which I found to work very poorly for discriminative feature optimization. The general idea of optimizing towards the expected value of an evaluation metric can be traced back to Povey and Woodland [27], which describes a method for the discriminative training of a speech recognition system through expected-WER minimization.

In order to perform high-dimensional feature weight optimization, Chiang et al. [9] implemented MIRA, which is a natural extension of Minimum Error Rate Training (MERT) [22] (in the same way Expected-BLEU tuning is a natural extension of 1-best BLEU with Powell's method). In the past, I spent a significant amount of time attempting integrate MIRA into our system without success. Note that the goal of both MIRA and Expected-BLEU tuning is identical: Find the weights that produce the maximal BLEU score on a tuning set. However, MIRA does this in a very roundabout way, by minimizing the *hinged loss* of an *oracle translation*, so it is very difficult to diagnose the problem when it doesn't perform well. Expected-BLEU tuning performs optimization in a very direct way, by actually maximizing the (expected-value of the) BLEU score.

Chapter 3

Analysis of Lexical Features

This chapter describes the translation features and experimental results.

3.1 Description of Translation Features

In the hierarchical decoding model there are two major feature types which model translation: the rule translation probabilities and the lexical smoothing scores.[1] Given a rule r with source phrase S and target phrase T, the rule probabilities $P_r(S|T)$ and $P_r(S|T)$ are simply the maximum likelihood estimates of the rule counts, i.e.:

$$P_r(T|S) = \frac{C_r(S,T)}{\sum_{T'} C_r(S,T')} \tag{3.1}$$

$$P_r(S|T) = \frac{C_r(S,T)}{\sum_{S'} C_r(S',T)} \tag{3.2}$$

Where $C_r(S,T)$ is the total number of times the rule $S \rightarrow T$ is seen during rule extraction. Technically, a rule can actually be defined by more than its source and target phrases—in the current system, it is also uniquely defined by its left-hand-side non-terminal, word alignment, part-of-speech tags, and dependency structure.

[1] I alternative between using the singular and plural to describe the lexical smoothing score(s). Technically, there are actually two rule probabilities and two lexical smoothing scores—one in the forward direction, and one in the backward direction. However, historically, the rule probabilities have been kept separate while the lexical smoothing scores are multiplied together, so the exponent for each direction is not independently optimizable. This is discussed later in this section.

However, the source (or target) count[2] of a rule is always summed over all the rules with the same source non-terminal[3] and source phrase S (or target non-terminal and target phrase T).

Before defining the lexical smoothing scores, we must define lexical probabilities. Given a source word s and a target word t, the lexical probabilities $P_l(s|t)$ and $P_l(t|s)$ are simple maximum likelihood estimates over the lexical counts, i.e.:

$$P_l(t|s) = \frac{C_{lex}(s,t)}{\sum_{t'} C_{lex}(s,t')} \tag{3.3}$$

$$P_l(s|t) = \frac{C_{lex}(s,t)}{\sum_{s'} C_{lex}(s',t)} \tag{3.4}$$

Where the lexical count $C_{lex}(s,t)$ represents the number of times the source word s aligns to the target word t in the training data. In other words, the parallel training corpus is fed into a word alignment program, currently GIZA++, which produces an alignment matrix for each parallel source/target sentence pair. Each filled entry in this matrix is called an *alignment link*. If a source (target) word does not align to any target (source) words, an alignment link is added from that word to $NULL$. These are represented as $s \rightarrow NULL$ and $NULL \rightarrow t$ respectively. The lexical counts $C_{lex}(s,t)$ are thus generated by counting the alignment links in all of the alignment matrices in the training.

As mentioned above, these lexical probabilities are used within the lexical

[2]In case the terminology is unclear, the "source count" of a rule is the denominator in Equation 3.1.

[3]Although I used only a single non-terminal X when describing hierarchcial translation, there can actually be an arbitrary number of non-terminal types. However, the details are not important with respect to this thesis.

smoothing formulae, which are static features[4] of each translation rule r. The *forward* lexical smoothing formula represents the intuition that each target word should be a high probability translation of at least one source word. The *backward* lexical smoothing formula is the reverse of this, representing the intuition that each source word should be a high probability translation of at least one target word. The current system uses a formula known as "noisy *or*" [39].[5]

$$L_f(S,T) = \prod_{t \in T}(1 - \prod_{s \in S \cup NULL}(1 - P(t|s))) \quad (3.5)$$

$$L_b(S,T) = \prod_{s \in S}(1 - \prod_{t \in T \cup NULL}(1 - P(s|t))) \quad (3.6)$$

Note that the inner product iterates over *all* source (target) words and $NULL$, not just the ones that align to t (s). The reason for this is that word alignments have historically been stripped out of the translation rules, so it works even when the alignments are not available during the lexical smoothing calculation. If the alignments are kept, these alternative formulae may be used:

$$L_f(S,T) = \prod_{t \in T}(1 - \prod_{s \in A(S,t)}(1 - P(t|s))) \quad (3.7)$$

$$L_b(S,T) = \prod_{s \in S}(1 - \prod_{t \in A(T,s)}(1 - P(s|t))) \quad (3.8)$$

where $A(S,t)$ is the set of source words in $S \cup NULL$ that align to t, and vice versa for $A(T,s)$.

[4] A "static" feature is one which does not depend on where a rule is applied during decoding. Context-dependent lexical smoothing, which is described in Chapter 5, is a "dynamic" feature, because its value depends on surrounding words outside of the rule.

[5] The formula could also be termed a "probabilistic *or*" since it gives the probability that this source word is a translation of *any* target word.

Equations 3.5, 3.6, 3.7, and 3.8 are somewhat unintuitive when first examined. Let us first take a closer look at the inner product of Equation 3.5:

$$LI_f(S,t) = 1 - \prod_{s \in S \cup NULL} (1 - P(t|s)) \tag{3.9}$$

If $P(t|s)$ is high for *any* s, then $1 - P(t|s)$ will be low. So even if $P(t|s) = 0$ for all other s, the total product $\prod_{s \in S \cup NULL}(1 - P(t|s))$ will be low. Thus, the equation $1 - \prod_{s \in S \cup NULL}(1 - P(t|s))$ will be high. This represents the intuition that t only needs to be translated by *one* source word with high probability in order to be considered correct. Then, Equation 3.5 is the product of $LI_f(S,t)$ over all t in T. Since this is a product, if one $LI_f(S,t)$ value is low, then the entire product will be low. This represents the intuition that *all* target words have to be translated with high probability in order for the lexical smoothing score to be high. Of course, Equation 3.5 is only a product over the target, so it will not penalize extraneous source words. This is the reason why the backward formula Equation 3.6 is necessary.

A more intuitive version of the lexical smoothing formulae are used by Koehn's Pharaoh decoder [17]. The equations for these are:

$$LK_f(S,T) = \prod_{t \in T} \frac{\sum_{s \in A(S,t)} P(t|s)}{||A(S,t)||} \tag{3.10}$$

$$LK_b(S,T) = \prod_{s \in S} \frac{\sum_{s \in A(T,s)} P(s|t)}{||A(T,s)||} \tag{3.11}$$

In other words, for the forward equation, the inner formula is the the mean lexical probability between t and all the source words that align to t.

Three different experiments were run using three formula pairs described above, and all produced nearly identical decoding results. However, note that Equation 3.7/3.8 and Equation 3.10/3.11 are significantly faster to compute than Equation 3.5/3.6, since they only sum over the aligned links instead of the entire phrase. The standard lexical smoothing formulae are very inexpensive to compute so this is not a major factor, but the context-dependent lexical smoothing described in Chapter 5 can be somewhat costly, so we will revisit these formulae then.

Historically, the rule probabilities $P_r(T|S)$ and $P_r(S|T)$ have been used as separate features, while the forward and backward lexical smoothing formulae have been multiplied together to form a single feature, defined as:

$$L(S,T) = \sqrt{L_f(S,T) * L_b(S,T)} \tag{3.12}$$

$$\log(L(S,T)) = 0.5 \log L_f(S,T) + 0.5 \log L_b(S,T) \tag{3.13}$$

The effect is that the forward and backward lexical smoothing scores cannot receive separate exponents in optimization. Clearly, in a log-linear model these can be separated out into two features. An experiment was run with the forward and backward lexical smoothing formulae separated out, and the results were nearly identical to the baseline. However, in Chapter 5, I introduce the concept of context-dependent lexical smoothing, where the forward and backward formulae are always separated out.

3.2 Experimental Results

My focus on lexical features began after running a small series of experiments comparing the relative importance of the rule translation probabilities and the lexical smoothing scores in the current system. I ran three experiments, one where the rule probabilities were turned off, one where the lexical smoothing features were turned off, and one where both were turned off. The results of these experiments are presented in Table 3.1. Surprisingly, turning off the rule probabilities only degrades the results by about 1.5 BLEU, while turning off the lexical smoothing score degrades the results by about 3.5 BLEU. Since these are the two major features for modeling translation, we also see that turning them both off degrades the results by about 10 BLEU. Federico and Bertoldi [11] previously showed that the rule probabilities can be quantized into only 5 bits (i.e., 32 histogram bins) without degrading the BLEU scores, but they did not try turning them off completely.

	Chinese Newswire		
	Tune	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*
Baseline	**40.48**	**27.05**	**40.05**
No Rule Trans	39.16	25.64	38.49
No Lex Smooth	37.13	24.21	36.42
No Both	30.60	20.20	29.91

Table 3.1: In *No Rule Trans*, the rule probability features $P_r(T|S)$ and $P_r(S|T)$ were turned off. In *No Lex Smooth*, the lexical smoothing feature $L(S|T)$ was turned off. In *No Both*, both the lexical smoothing and rule probabilities were both turned off. There were no major features modeling translation. All conditions were re-optimized. The best result in each column is in bold font.

These results are surprising because word-based translation models fell out of favor years ago, being first replaced by phrase-based models and more recently,

hierarchical models. In fact, it is worthwhile to think about all of the things that are implicitly included in rule probabilities which lexical smoothing fails to account for: word order, redundancy, non-terminal placement, and more. In spite of this, I have shown that the lexical smoothing score is significantly more important[6] than the rule probabilities in a state-of-the-art hierarchical system.

In this thesis, I do not suggest a return to word-based translation, but instead present novel methods for modifying the lexical probabilities that result in significant improvements to the current system. However, before describing the new features, I will first describe *why* I believe lexical smoothing is a more useful feature than translation probabilities.

In short, I believe that that this result can be explained by the accuracy of the probability estimation. In other words, the lexical translations are seen *many* more times on average than the rule translations, so the lexical probability MLEs are more accurate than the rule translation MLEs. In Table 3.2, *joint count percentiles* are compared for translation rules and lexical translations. As an example, for the lexical translation $hombre \rightarrow man$, imagine that the forward lexical probability is $P(man|hombre) = \frac{C_{lex}(hombre,man)}{C_{lex}(hombre)} = \frac{108}{132}$. In this case, the *joint count* is 108, which is equivalent to the number of times the rule $hombre \rightarrow man$ was seen in the training. So, Table 3.2 shows that 30% of rules used in the 1-best Arabic newswire output have a joint count of less than 3, while 30% of the lexical translations have a joint count less than 14711. The difference is striking. In fact, it turns out that

[6]Here, "important" refers purely to empirical importance, i.e., BLEU score. I am not making any statements about the philosophical importance of any feature.

only 0.5% of lexical translations used in the 1-best output have a joint count of 1, and only 1.3% have a joint count of less than 10. On the other hand, 21% of rules used in the 1-best output have a joint count of 1, and 44% have a joint count of less than 10.

Percentile	**Rule Count**	**Lexical Count**
0%	1	1
10%	1	777
20%	1	4336
30%	3	14711
40%	7	47134
50%	16	123317
60%	44	392951
70%	166	2531992
80%	920	5830785
90%	7769	40941610
100%	5788594	40941610

Table 3.2: Percentiles for rule joint counts (e.g., $C_r(S,T)$) and lexical joint counts (i.e., $C_{lex}(s,t)$), from the 1-best output on Arabic newswire. For example, out of all the rules used in the Arabic newswire 1-best output, 40% of them had a joint count less than 7. This *does* count duplicates, meaning if a rule was seen 10 times in the output, it was counted as 10 unique instances.

So, we know that a large percentage of the rules used are only seen a small number of times, which makes the rule probability MLEs unreliable. So why not just try to fix the *rule* probabilities? In order to better answer this, I will take a short detour to discuss language models. The purpose of an n-gram LM is to model the probability $P(w_n|w_1, w_2, ..., w_{n-1})$ that the word w_n follows the $n-1$ previous words $w_1, w_2, ..., w_{n-1}$. The $(n-1)$-gram $w_1, w_2, ..., w_{n-1}$ is referred to as the *context* of an n-gram. As n grows larger, the LM uses more context and thus more accurately models the language, but simultaneously the n-gram counts become smaller and thus less reliable. Imagine that the 5-gram LM was simply defined as the maximum

likelihood estimate:

$$P(w_5|w_1,w_2,w_3,w_4) = \frac{C(w_1,w_2,w_3,w_4,w_5)}{\sum_{w'} C(w_1,w_2,w_3,w_4,w')} \quad (3.14)$$

When computing the LM score on an arbitrary test sentence, many of the 5-grams will be unseen and will get an artificially low probability near zero.[7] However, if a 5-gram and its 4-gram context are only seen once, then this 5-gram will get an artificially high probability of one.

In reality, LM probabilities are *not* computed as simple MLEs, and instead they use a sophisticated *backoff model*. In a standard backoff model, the smoothed 5-gram probability $P(w_5|w_1,w_2,w_3,w_4)$ is created by interpolating the MLE with the 4-gram probability $P(w_5|w_2,w_3,w_4)$[8], where the interpolation weight is based on how much the MLE is "trusted." Generally, the more times the context w_1,w_2,w_3,w_4 has been seen, the more the 5-gram MLE is trusted.

Coming back to translation rules, the problem is that it is not obvious how to smoothing the rule probabilities in a way analogous to the language model. I do not mean to imply that it is an *impossible* problem or that there has been no success in it, but simply that there are no *widely accepted* methods for robust rule probabilities smoothing (as there are for language model probability smoothing). Therefore, in this thesis I take an alternate approach and treat the lexical probabilities as

[7] Technically, the probability should be *exactly* zero, but it is rarely a good idea to use zero probabilities in a multiplicative model.

[8] Of course, the smoothed 4-gram probability $P(w_5|w_2,w_3,w_4)$ is recursively computed using the 4-gram MLE and the 3-gram probability $P(w_5|w_3,w_4)$, and so on.

being of primary importance when modeling translation[9], and largely ignore the rule probabilities.

It now follows that I turn the focus towards improving the lexical probabilities. In the remaining chapters I will present a number of methods for improving these probabilities, and demonstrate that some of these methods can produce a significant gain in BLEU.

[9]Since *currently*, in our system, they are.

Chapter 4

Lexical Probability Biasing

Lexical probability biasing is the process of effectively duplicating some *bias data*[1] n times and then adding it to the original training data before lexical probability extraction. In general, the contents of the bias data is dependent on the test document being decoded, which is why it is not just added to the training to begin with. Formally, the formula used here for biased lexical probabilities is:

$$P_{bias}(t|s) = \frac{C_T(s,t) + n * C_B(s,t)}{C_T(s) + n * C_B(s)} \tag{4.1}$$

where C_T is the count from the original training and C_B is the count from the bias data. The biased lexical probabilities are then used in the standard lexical smoothing formula.

I experimented with two very different sources of bias data. The first method, called *semantic biasing*, biases each test document D with parallel training documents that are semantically similar to D. The second method, called *self-adaptation*, biases the test data with its *own* aligned output from a first-pass decoding run. The details for both methods are described in the following sections.

[1] I say that the data is *effectively* duplicated because in practice the counts are interpolated at run time, which is mathematically equivalent but much more efficient.

4.1 Lexical Semantic Biasing

In the lexical semantic biasing procedure, for each test document D the system selects m semantically similar documents from the training as bias documents. Note that this means the biased lexical counts are different for each test document, hence the reason for runtime interpolation of the biased lexical probabilities. The documents were selected using a cross-lingual-information-retrieval (CLIR) system built by researchers at BBN [37], which uses the source side of the test documents to select documents in the target language. I also tried using monolingual-IR (MIR) for document selection, where the source side of the test set was used to select documents in the source language. However, for the purposes of this feature these two systems do not produce appreciably different results.

All of the documents selected are already present in the training, so the biasing procedure is simply weighting them more than the other documents by the duplication factor n. Intuitively, when translating an article about the Mars probe, for example, other articles about NASA and space travel will generally have more accurate lexical translations than a random article from the training.

Table 4.1 reports results on several different experimental conditions: CLIR vs. MIR, fixed vs. variable duplication factor, and inclusion vs. exclusion of function words. The regular CLIR condition gets a gain of 0.14 BLEU on Tune and 0.38 BLEU on Test 2, but it loses 0.19 BLEU on Test 1.

	Chinese Newswire		
	Tune	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*
Baseline	38.73	**26.76**	39.22
MIR	38.84	26.76	39.55
CLIR	**38.87**	26.57	**39.60**
CLIR - no func words	38.75	26.35	39.33
CLIR - 1/n weighting	38.75	26.69	39.45

Table 4.1: Results for lexical semantic biasing. The top 10 training documents closest to each test document were retrieved using CLIR/MIR, and the lexical counts were biased at runtime using these documents. $1/n$ weighting means that the n^{th} closest document was given a prior weight of $1/n$ (which is still multiplied by the interpolation weight). Weights from 100 to 100,000 were tried, and the best results are presented here.

4.2 Lexical Self-Adaptation

Self-adaptation (also known as self-training) is the process of using first-pass MT output to modify the models in a second pass of decoding. In order to perform lexical probability self-adaptation (SA), a word-level alignment is required between the source sentence and MT output. The normal procedure is to strip out word alignments in translation rules, since keeping the alignments causes rule fragmentation and degrades the results slightly. I made a slight modification to the rule extraction procedure such that word alignments were kept during rule extraction, but then rules that differed only by alignment were merged and the most frequent alignment was kept.

I performed the self-adaptation at four different levels:

1. Sentence level—Bias each test sentence s with the first-pass output of just s itself.

2. Document level—Bias each test sentence s with the output of the test document D that contains s. These documents are generally 10-20 sentences long.

3. Test set level—Bias each test sentence s with the output of the the test set containing s. The test sets contain 1000-2000 sentences.

4. Dev set level—Bias each test sentence s with the output of the entire dev set, which contains three test sets (one tuning and two validation). The entire dev set contains about 6000 sentences.

More complicated types of biasing are possible, such as biasing each sentence with its own document plus the ten most similar documents in the dev set (based on CLIR). However, the more complicated methods did not show a gain over the four levels described above, so I will not report on those results.

The results for lexical probability self-adaptation are presented in Table 4.2. Note that self-adaptation can be run for an arbitrary number of iterations, although in practice is it unlikely that anything over two rounds of self-adaptation (three decodes total) will help. Somewhat unexpectedly, dev set level SA performs best, gaining 0.34, 0.23, and 0.52 BLEU on Tune, Test 1, and Test 2, respectively. Performing a second round of self-adaptation using the results from the first round does slightly better. However, since the gains are relatively small, and the procedure is quite expensive (since everything must be run twice), self-adaptation is not used in the standard system.

It is not immediately obvious what is producing this gain, but I propose four possible mechanisms:

	Chinese Newswire		
	Tune	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*
Baseline	38.73	26.76	39.22
Sentence SA	39.02	26.75	39.18
Document SA	38.97	26.96	39.60
Document SA - Round 2	39.08	26.57	39.55
Test Set SA	39.03	26.93	39.41
Dev Set SA	39.07	**26.99**	39.74
Dev Set SA - Round 2	**39.21**	26.97	**39.97**
Dev Set SA - Round 3	38.98	26.97	39.88

Table 4.2: Results for lexical smoothing self-adaptation (SA). In all cases, the lexical counts from the MT output were interpolated with the lexical counts from the training data with a weight of 100,000. The four types of self-adaptation are described above. "Iter 2" means that the lexical counts were computed from the first round of self-adaptation rather than the baseline.

1. **Language model feedback**—In the standard decoding process, a 3-gram LM is used for decoding, thus producing a 1000-best list. This list is then rescored with a 5-gram LM to produce the final MT output. When the rescored output is fed back into the decoder a significant gain is expected in the 3-gram decoding results because this second 3-gram result benefits from the more powerful 5-gram LM. The effect of this feedback can clearly be seen in Table 4.3, which shows a significant gain under all conditions *before* 5-gram rescoring. Since the 5-gram output is new information to the decoder, it is possible that the model is changing enough to get better results even after rescoring. However, if this were the major motivating factor, the sentence-level SA would be expected to do the *best*, because LM rescoring is done at the sentence level. Instead, sentence-level SA is the only condition to do *worse* than the baseline after rescoring, so this is unlikely to be the major factor.

	Chinese Newswire		
	Tune	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*
Baseline - Prelim	36.81	25.99	37.39
Sentence SA - Prelim	**37.76**	**26.23**	**38.23**
Dev Set SA - Prelim	37.35	26.19	37.76

Table 4.3: Self-adaptation results *before* 5-gram rescoring. These results should *not* be interpreted as a genuine gain, but are instead presented to show the effect of feeding 5-gram rescored MT output back into a 3-gram decoding system. The gains would be expected to shrink significantly after 5-gram rescoring, and in fact "Sentence SA" gains *nothing* over the baseline in the final results.

2. **Implicit lexical context**—The lexical probabilities are context free, but the overall MT output is clearly not—it is produced using the language model, translation rules, and many other features. Thus, the lexical translations seen in the final MT output are *implicitly* context dependent, meaning that they rely on the context provided by independent features. It is reasonable to believe that biasing the context free lexical probabilities with these context dependent translations would be beneficial. However, since translation is performed on the sentence level, if this were the major motivating factor then sentence-level SA should perform very well, which it does not.

3. **Lexical translation voting**—Under the well-known "one sense per discourse" rule of thumb, a source word that is seen multiple times within the same document will be expected to have a similar translation in all cases. Thus, if a word is seen three times within a document, and the translation is correct twice and incorrect once, the correct translations will essentially "outvote" the incorrect translation, and the incorrect translation may be fixed in the second pass of

decoding.

There are two caveats to this. First, in the preceding example, the biased lexical probability is 0.66, so if the regular lexical probability is greater than 0.66, the biasing will actually *reduce* its probability. However, 0.66 is a very high lexical probability and if $P(t|s) \geq 0.66$ then the system is very unlikely to translate s to anything other than t, in any context. Second, if more sentences in the document contain more incorrect translations of s than correct, the biasing could transform a correct translation into an incorrect one. However, the MT system generally produces mostly correct translations. Empirically, the document level self-adaptation gives a moderate gain and manual analysis shows that this type of voting does occur in the test set, so this at least partially explains the gain from lexical self-adaptation.

4. **Implicit smoothing**—The distribution of lexical probabilities in training data is very different from the distribution of lexical probabilities in MT output. Specifically, common source words tend to have many more unique translations and a more "spread out" probability mass in human-translated parallel text compared to MT output. Table 4.4 shows the lexical probability distribution of the Chinese word *shuo* ("said") calculated on both the MT output of the dev sets and the GIZA aligned human references. Note that the MT output has many fewer unique translations and a significantly higher probability for the most common translation ("said").

   Part of the reason for this difference in translation distribution is noisy training

GIZA Alignment		**MT Output**	
Num Unique Trans	56	Num Unique Trans	22
English Trans	**P(t|s)**	**English Trans**	**P(t|s)**
said	0.51	said	0.62
that	0.29	that	0.25
NULL	0.06	NULL	0.03
says	0.03	:	0.01
stated	0.02	today	0.01

Table 4.4: 5 most common translations for *shuo* from the GIZA aligned dev set and MT output of the dev set. Note that these two tables were computed on the exact same Chinese source data. Also remember that lexical probabilities are computed over alignment links, so the alignment "*shuo* → said that" will modify both $P(said|shuo)$ and $P(that|shuo)$. This illustrates a clear smoothing effect, where uncommon translations like "stated" are translated as common translations like "said."

data—we know that a portion of the parallel corpus is mis-translated, mis-segmented, or mis-aligned. However, since these errors are mostly random, the lexical probabilities of erroneous translations will generally be quite low, especially for common words such as *shuo*. I performed an experiment where all lexical probabilities below 10^{-4} were thrown out and the probabilities were distributed among the remaining translations, but the results were almost identical to the baseline.

The bigger reason for the difference in the distributions is likely an effect of normal human language production. Translating *shuo* as "said" would probably be correct in almost all cases, but it would sound awkward and robotic, so humans often translate it as "stated," "added," or "explained." This type of random (but linguistically valid) variation in the training data may actually be harmful for MT, but it is difficult to model explicitly. In the case of "said," the self-adaptation makes the most probable translation even

more probable, and lowers the probability of the synonymous translations, which intuitively is a good outcome—at least in this case.

In this chapter, I presented two methods of *biasing* the lexical probabilities, where *biasing* means to modify a probability distribution based on the document being decoded. The first method, lexical semantic biasing, extracts biased counts from the n most similar training documents to the test document being decoded, where similarity is computing using a cross-language information retrieval (CLIR) system. The second method, lexical self-adaptation, biases each document with its own output from a first-pass decode.

Chapter 5

Context-dependent Lexical Smoothing (CLS)

It is well-established that contextualization is beneficial to the accuracy of both human and machine translations, and—by their very nature—hierarchical translation rules utilize far more context than the lexical probabilities. However, as shown in Chapter 3, the rule translation probabilities are far less important to the current system than the lexical smoothing score, which is based entirely on context-free lexical probabilities. I believe the reason for this is that the rule probabilities are unreliable, due to the inherent sparsity involved and the lack of a backoff model. The lexical probabilities, on the other hand, are seen far more times than the translation rules on average, and so the maximum likelihood estimates are going to be reliable. Thus, an attempt can be made to improve the lexical probabilities by adding context, *but only when the context-dependent lexical translations are seen enough to be reliable.*

This chapter describes the intuition and formal model behind context-sensitive lexical smoothing (CLS). In addition, my approach to weight estimation is presented and experimental results are reported.

5.1 CLS Model

The context-dependent lexical probabilities are a natural extension of the context-free lexical probabilities. For example, one context type is *previous word*, formally written as $P(t|s_i, s_{i-1})$[1]. The MLE of the context-dependent lexical counts is:

$$P(t|s_i, s_{i-1}) = \frac{C(s_i, t, s_{i-1})}{\sum_{t'} C(s_i, t', s_{i-1})} \tag{5.1}$$

The context-dependent counts are computed over the alignment links in the same way as the context-free counts, except that only the actual context is added in. For example, consider the three word sentence:

yo hablo . → i talk .

For the *previous word* model, this would result in the following three context-dependent lexical counts: C(yo,i,<s>), C(hablo,talk,yo), and C(.,.,hablo). Note that the "sentence begin" and "sentence end" markers are added, as is done for language modeling.

I experimented with the following types of lexical contexts:

- Context-free: $P(t|s_i)$
- Part-of-speech: $P(t|s_i, POS(s_i))$
- Previous word: $P(t|s_i, s_{i-1})$
- Previous two words: $P(t|s_i, s_{i-1}, s_{i-2})$

[1]Note that this is *not* the same as the translation rule $s_{i-1}s_i \rightarrow t$, which implies that t is a translation of both s_i and s_{i-1}. The context-dependent lexical probability $P(t|s_i, s_{i-1})$ says that t is a translation of s_i, and s_i is preceded by s_{i-1}. This implies nothing about the relationship between t and s_{i-1}.

- Previous part-of-speech: $P(t|s_i, POS(s_{i-1}))$
- Next word: $P(t|s_i, s_{i+1})$
- Next two words: $P(t|s_i, s_{i+1}, s_{i+2})$
- Next part-of-speech: $P(t|s_i, POS(s_{i+1}))$
- Previous word and next word: $P(t|s_i, s_{i-1}, s_{i+1})$

In the previous example, the context models are only defined for *forward* lexical probabilities. However, all of the above models can also be used in the backward direction, e.g., $P(s|t_{i-1}, t_{i-2})$. There is one caveat when using backward context-dependent lexical probabilities, which will be discussed later. First I will describe how the context models are used to modify the lexical smoothing score.

It is not appropriate to simply replace the context-free probability model with a higher-order model, because that would result in the same estimation problems seen in the translation rules. Additionally, it would be highly desirable to be able to factor in several types of context simultaneously. The solution is, then, to use a formula that can interpolate an arbitrary number of lexical probability models to create a *interpolated lexical probability*. The formula used here does not actually use the context-dependent lexical probabilities directly; instead it interpolates the lexical *counts*. The formula for interpolating N lexical probability models is:

$$P_I(t|s) = \frac{\sum_{i=1}^{N} w_i C(s, t, c_i)}{\sum_{i=1}^{N} \sum_{t'} w_i C(s, t', c_i)} \tag{5.2}$$

where c_i is the context of the i^{th} model, and w_i is the weight for the i^{th} model.

The first thing to note is that this formula is a natural extension of the formula

given for lexical probability biasing, Equation 4.1. As was mentioned in Chapter 4, interpolating the counts in this way is equivalent to duplicating the training—which admittedly has less theoretical grounding when interpolating *different* models (as is done here) compared to biasing the *same* model (e.g., lexical semantic biasing). However, it does immediately follow that the interpolated lexical probability distributions are well formed:

$$\begin{aligned}
\sum_t P_I(t|s) &= \sum_t \frac{\sum_{i=1}^N w_i C(s,t,c_i)}{\sum_{i=1}^N \sum_{t'} w_i C(s,t',c_i)} \\
&= \frac{\sum_t \sum_{i=1}^N w_i C(s,t,c_i)}{\sum_{i=1}^N \sum_{t'} w_i C(s,t',c_i)} \\
&= \frac{\sum_{i=1}^N \sum_t w_i C(s,t,c_i)}{\sum_{i=1}^N \sum_{t'} w_i C(s,t',c_i)} \\
&= 1 \qquad (5.3)
\end{aligned}$$

Secondly, the count-based interpolation implicitly shifts the probability mass towards contexts that have been seen more times, independent of the actual probability. This represents the intuition that the more times an event has been seen, the more accurate the estimate of the probability is believed to be.

The alternative to Equation 5.2 is, of course, to interpolate the probabilities. A *fixed-weight* probability interpolation model would not accurately represent our trust model, since a count of $\frac{1}{2}$ would contribute the same probability mass as $\frac{500}{1000}$. Language model smoothing formulae like Kneser-Ney (KN) [16] and Witten-Bell (WB) [36] use a *variable-weight* probability interpolation model that follows the same intuition as was described previously—the more times an estimate has been

seen, the more trustworthy it is. However, in the language model, there is a clear ordering between models, e.g., 3-gram backs off to 2-gram, 2-gram backs off to 1-gram. For the interpolated lexical probability, it is necessary to interpolate an arbitrary number of context-dependent lexical models that do not necessarily have such an ordering, and there is no clear way to do this with WB or KN. Additionally, the KN and WB formulae both make assumptions about language modeling that might not be true about translation modeling, such as the number of *unique* words that follow a given context being important when computing the backoff. However, I still ran experiments comparing WB and count-based interpolation using only two models, *context-free* and *previous word*, and the count-based interpolation significantly outperformed WB smoothing.

Earlier, I mentioned a caveat when performing context lexical smoothing in the backward direction. The issue is that certain types of context are not available for words in certain positions. Specifically, the *previous word* context is not available for the target word in a rule, and the *next word* context is not available for the last word, due to the inherent constraints of bottom-up decoding. Imagine the decoder is applying the hierarchical rule "X : el coche $X \rightarrow$ NP : the ADJ car". In this example, the probabilities $P(\text{coche}|\text{car}, t_{i+1})$ and $P(\text{el}|\text{the}, t_{i-1})$ cannot be computed, so the joint and target counts for these are simply treated as 0. However, note that the decode *can* look "inside" of the target non-terminal ADJ, because the first two and last two words of each non-terminal must be available for trigram language modeling. Assuming that the rule filling ADJ is "X : rojo $\rightarrow$ ADJ : red", then the backwards CLS score would be computed using the probabilities $P(\text{coche}|\text{car})$ and

$P(\text{coche}|\text{car}, \text{red})$, as well as $P(\text{el}|\text{the})$ and $P(\text{el}|\text{the}, \text{red})$. In rescoring, backwards smoothing scores can be computed exactly for each hypothesis, since the entire sentence is known at that time. However, there was no additional gain for doing this.

5.2 Weight estimation

For each lexical probability model, it is necessary to estimate a fixed count-based interpolation weight w_i. With only two lexical probability models, it is possible to simply perform a binary search on the weight, but it is highly desirable to be able to use an arbitrary number of models. Since the lexical smoothing formula is non-linear, attempting to optimize the weights to directly maximize BLEU scores would be difficult. Instead, the weights can be optimized to maximize a *correlate* of BLEU, specifically maximizing the lexical likelihood of the tuning set.[2]

Computing the lexical likelihood is equivalent to computing the lexical smoothing score on a set of aligned sentences. I use Equations 3.7/3.8 to compute the lexical smoothing over the alignment links, since the scores are being computed on sentences instead of rules. Now imagine that there are two arbitrary lexical probability distributions A and B—for example, A is context-free and B is interpolated context-dependent, or A is trained on UN data while B is trained on newswire. The idea is that whichever lexical probability distribution has a higher lexical likelihood on some withheld parallel data will *generally* be able to translate the data more ac-

[2]This can either be some withheld set of training data, or the actual tuning set GIZA aligned to its references.

curately when used in the lexical smoothing feature. Since the lexical likelihood can be computed very quickly, optimization can be performed using a generic gradient descent algorithm with a numerically approximated derivative, where the objective function is the lexical likelihood of the tuning set.

Empirically, I found that the results are very *in*sensitive to the weights, and are thus easier to simply estimate manually. By this I mean that changing an interpolation weight by a factor of 10 or even 100 often does not change either the BLEU or lexical likelihood scores by any appreciable amount, so it is easy to manually find an "optimal" set of weights within a small number of tries.

To get an intuitive sense of what good interpolation weights should be, we can look at the lexical counts themselves. The median source count for the *context-free*, *previous word*, and *next word* models are 87, 0.57, and 0.56 respectively.[3] This means that for a random interpolated lexical probability, it would be expected that the *context-free* source count (i.e., the denominator in $P(t|s)$) to be about $87/0.57 \approx 87/0.56 \approx 143$ times larger than the *previous word* or *next word* source count. So if the *previous word* and *next word* models were given an interpolation weight of 143 and *context-free* was given a weight of 1, then each model would be expected to contribute about $\frac{1}{3}rd$ of the probability mass in the interpolated probability. Of course, if a context model has a higher or lower count proportional to the context-free model, it will contribute more or less to the interpolated probability, which is the exact purpose of the algorithm. It is also reasonable to believe that

[3]The average source counts are non-integral and less than 1 due to the corpus weighting procedure.

the context models should, on average, contribute more probability mass than the context-free model and therefore have a weight significantly greater than 143. The empirical results, which I will present next, support this notion.

5.3 Experimental Results

The results for a number of different context lexical smoothing conditions are presented in Table 5.1. The condition "3 models FW+BW" produces the best results with the fewest number of models. This condition includes the models *previous word*, *next word*, and *context-free*, and it uses them in both the forward and backward direction. This condition gets a BLEU gain of 0.47 on Tune, 0.28 on Test 1, and 0.60 on Test 2. A fixed, "manually optimized" interpolation weight of 1000 is used for *previous word* and *next word* (with a weight 1 for *context-free*). However, it should again be noted that the weights are very flexible, and results were similar using weights of 100 or 10,000.

Using the additional context models *previous two words*, *next two words*, and *previous word and next word* did not help, nor did computing the true backwards lexical smoothing score during n-best rescoring. Although I do not report the results here, many of the other context models such as *previous two words* did help on their own (i.e., interpolated with just *context free*, compared to the baseline), but did not help on top of the "3 models FW+BW" condition. POS-based models did not help at all, either on their own or on top of "3 models FW+BW."

It is also interesting that using context in just the forward direction and com-

	Chinese Newswire		
	Tune	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*
Baseline	38.73	26.76	39.22
3 models FW	37.09	26.97	39.75
6 models FW	38.95	27.09	39.59
3 models BW	39.16	**27.11**	39.43
3 FW+BW	39.20	27.04	**39.82**
3 FW+BW, BW rescore	**39.25**	26.82	39.75
3 FW+BW, Aligned	39.04	26.87	39.82
3 FW+BW, Doc SA	**39.36**	**27.30**	39.85
3 FW+BW, Dev SA	39.36	27.20	**40.01**

Table 5.1: The "3 models" are *previous word*, *next word*, and *context-free*, with weights of 1000, 1000, and 1 respectively. "6 models" adds *previous two words*, *next two words*, and *previous word and next word*, all with weights of 100,000. "BW rescore" means that the true backwards score is computed on each of the final n-best hypotheses, as described above. The "Aligned" conditioned means that the lexical smoothing is only computed over the alignment links, as in Equations 3.7/3.8. "Doc SA" and "Dev SA" add document-level and dev set-level self-adaptation on top of context lexical smoothing.

puting the lexical smoothing score over the alignment links only seem to hurt a small amount compared to the "3 models FW+BW" condition. As is discuss in the next section, both of these conditions speed up the computation significantly, and so they are useful to use in some cases.

Finally, using self-adaptation gets a small gain over the "3 models FW+BW" condition, but since it is quite expensive (every decoding experiment must be run twice), it is not used as part of our standard system. However, since the gain for self-adaptation is smaller than previously reported but still present, it is reasonable to conclude that local context is *part of* what makes lexical self-adaptation work, but not the *only* factor.

The final results on Arabic are presented in Table 5.2. There is a similar gain

of 0.5-0.6 BLEU on Arabic as on Chinese.

In this chapter, I presented a novel method of contextualizing the lexical translations while still ensuring that the lexical probabilities are well-estimated. This process consists of defining an arbitrary number of context types, computing the context-dependent counts from the training, and interpolating all of the context types together as a single MLE using a *count-based* interpolation formula.

	Chinese Newswire			**Arabic Newswire**		
	Tune	**Test 1**	**Test 2**	**Tune**	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*	*BLEU*	*BLEU*	*BLEU*
Baseline	38.73	26.76	39.22	56.46	52.60	54.99
3 FW+BW	**39.20**	**27.04**	**39.82**	**56.80**	**53.11**	**55.60**

Table 5.2: CLS results on both Chinese and Arabic.

5.4 Implementation Issues

Because the context-dependent lexical smoothing scores must be computed on the fly, there are memory and speed issues to contend with when the features are used in decoding. Memory usage can be significantly reduced by filtering out unusable counts during load-time. In our hierarchical decoding system, the general procedure is to run a batch of parallel decoding jobs with 15 sentences in each job. The rule file is also filtered on each of these sentence sets and loaded into memory. Thus, a lexical count only needs to be loaded into memory if it could possibly be applied to one of the 15 sentences in the set using one of the rules in the filtered rule file.

Formally, define a context-dependent lexical count as a source-target n-gram

pair $C = \langle s_0 s_1 ... s_n, t_0 t_1 ... t_m \rangle$. Note that for forward context, $n = 2$ and $m = 1$ (e.g., $C = \langle hombre\ el, man \rangle$) and for backward context, $n = 1$ and $m = 2$. To perform the filtering, only load C into memory if (1) $s_0 s_1 ... s_n$ is an n-gram of one of the sentences being decoded by the current job[4], (2) each of the words $\{t_0, t_1, ..., t_m\}$ appears in *any* (but not necessarily the same) rule in the filtered rule file, and (3) the words s_0 and t_0 appear in the *same* rule in the filtered rule file. I believe that this filtering is as close to optimal as is reasonably possible without using an exorbitant amount of computation to check the validity of each count C. This filtering is *exact*, meaning that it only filters out counts which cannot possibly be applied. *Heuristic* filtering can also be used, where lexical translations $s \rightarrow t$ are filtered out if $P(t|s) < \gamma$ for some small γ such as 10^{-5}.

The CLS formula it is quite a bit more expensive to compute than any of the other decoding features, so it can slow down decoding by a significant factor. For a rule containing n source terminals and m target terminals with k context models, the forward and backward lexical smoothing scores both require roughly $2 * n * m * k$ hash lookups. This is not a huge amount of computation—for an average rule where $n = 4$, $m = 4$, and $k = 3$, this is less than 100 operations—but keep in mind that the *second* most expensive feature is the language model, which only requires a handful of hash lookups. Most other features have been pre-computed. Empirically, there was a total slowdown of $\sim$2.5 in the initial experiments when forward and backward context-dependent lexical smoothing were used. This increase in runtime can be

[4]Since s_1 may actually be s_0's *previous* word, the string may need to be re-ordered before this check is performed. Alternatively, just check whether $\{s_0, s_1, ..., s_m\}$ are all words in the same sentence, which is slightly less optimal but cleaner.

mitigated in a number of ways:

1. **Only use context in the forward direction**—This will immediately cut the runtime in half, since the lexical smoothing score only needs to be computed on the fly in one direction. The context-free lexical smoothing score in the backward direction can be pre-computed for each rule. I found that using context in just the forward direction does not cause a degradation in Arabic, but does cause a slight degradation in Chinese.

2. **Compute the lexical smoothing over alignment links**—The default lexical smoothing formula, shown in Equations 3.5/3.6, computes the inside product over every terminal in the rule. The alternative formula, shown in Equations 3.7/3.8, only computes the inside product over aligned words. This reduces the number of hash lookups by several factors. Again, I found that this does not cause a degradation in Arabic, but does cause a slight degradation in Chinese.

3. **Cache the lexical smoothing score**—In the forward direction, when a given rule r is applied to the same source span in the same sentence, the results can be cached. In hierarchical decoding the same rule is often applied to the same span many times due to the nature of the language model computation, so this type of caching can save a significant amount of computation. In the backward direction the context-dependent lexical smoothing score cannot be efficiently cached.

4. **Pre-compute the content-dependent lexical probabilities**—In the for-

ward direction, the interpolated context-dependent probabilities can be precomputed at the start of a decode for each source word in the sentence being decoded. For a given source word s_i, there is only a single s_{i-1} and s_{i+1} with respect to the current sentence, so the interpolated probability can be computed for *every* t that s_i translates to in the lexical probability table. This saves computation because s_i will align to a particular t in many rules, so instead of having to perform $2k$ hash lookups (where k is the number of context models) to compute $P_I(t|s)$ in each rule, it is only necessary to compute $P_I(t|s)$ once. This removes some amount of flexibility so it is not currently done in the hierarchical system, but it is done in the real-time system.

Using these methods, the slowdown from CLS was reduced from $\sim$2.5x to $\sim$1.3x.

Chapter 6

Discriminative Lexical Features

As described in Section 3.1, the lexical probabilities $P(t|s)$ and $P(s|t)$ are estimated as maximum likelihood estimates over the aligned training sentences—this is an example of a *generative* model. By contrast, a *discriminative* model *directly* searches for the set of posterior probabilities $P(t|s)$ that maximize BLEU (or some other measure of accuracy) on some tuning set. In practice, we do not literally modify the probability $P(t|s)$, but instead give specific $s \rightarrow t$ word pairs such as *hombre* $\rightarrow$ *man* a binary feature, which is set to 1.0 if that word pair seen in the rule and 0.0 otherwise. The weight for this feature (a positive or negative value) is optimized along with all of the other weights in the log-linear translation model. Section 6.2 describes feature types, feature selection, and results related to discriminative training.

In the above example, each source-target word pair receives its own feature bin. This means that now *thousands* of feature weights must be optimized simultaneously instead of the normal $\sim$12. However, our long-standing optimization algorithm, 1-best optimization with Powell's method [28], becomes unreliable when more than a few dozen features are used.

Section 6.1 describes an alternate optimization method called Expected-BLEU tuning which can robustly maximize BLEU with tens of thousands of features. Note

that this section only describes optimization towards BLEU, but in fact the methods described here could be applied to any MT metric that has a differentiable expected value with respect to the decoding weights. Expected-TER [29] tuning can be implemented quite easily, and Expected-METEOR [1] would also likely be possible, although these are not explored here.

6.1 Expected-BLEU Tuning

Both 1-best and Expected-BLEU tuning are types of n-best optimization, meaning that for each of the S set sentences, the decoder generates N best unique hypotheses along with their decoding scores. The BLEU statistics are pre-computed for each hypothesis. The goal of both methods is to maximize the BLEU score on this n-best list, with the difference being that 1-best optimization attempts to maximize 1-best BLEU while Expected-BLEU tuning attempts to maximize the expected value of BLEU. Since 1-best BLEU is not a continuous function with respect to the decoding weights, the gradient cannot be taken. Thus, it is not possible to perform true gradient-based optimization and instead we use a line-search-like algorithm known as Powell's method [28]. I have empirically found this optimization procedure to be completely unreliable when more than a few dozen features are used. Since the expected value of BLEU *is* continuous with respect to the weights, a symbolic gradient can be computed and used with a generic gradient-based optimization algorithm such as L-BFGS [19]. I have found this to perform very well with tens of thousands of features *and* to be as robust as 1-best optimization

when only a small number of features are used. I will now formally describe both 1-best and Expected-BLEU optimization.

Assume that the n-best list consists of S test sentences $s = 1, ..., S$ which each have N hypotheses $n = 1, ..., N$.[1] For each hypothesis there is a feature vector of log-linear decoding scores F_{sk} and a set of BLEU statistics which will be defined later.

For a given set of weights w, define the log decoding score of the k^{th} hypothesis of the s^{th} test sentence as:

$$L_{sk}(w) = F_{sk} \bullet w \tag{6.1}$$

The 1-best hypothesis with respect to a given w and s is then simply the hypothesis with the highest decoding score, e.g. $\text{argmax}_k L_{sk}(w)$. 1-best optimization attempts to find the set of weights w that maximizes the 1-best BLEU score of the n-best list, with BLEU defined as:

$$BLEU(w) = \left(\prod_{i=1}^{4} \frac{M^{(i)}(w)}{K^{(i)}(w)} \right)^{1/4} * \min(1.0, e^{1-R(w)/H(w)}) \tag{6.2}$$

where the 10 terms $\{M^1(w), ..., M^4(w), K^1(w), ...K^4(w), R(w), H(w)\}$ are the *aggregate* BLEU statistics of the set of 1-best hypotheses. [2]

These 10 terms are pre-computed for each hypothesis, but since BLEU is

[1] Technically, very short test sentences will sometimes have fewer than N hypotheses available, but this is a formality we leave out for clarity.

[2] $M^{(i)}$ is the number of matching i-grams, $K^{(i)}$ is the number of hypothesis i-grams, R is the reference length, and H is the hypothesis length.

computed on a *set* of hypotheses, it is necessary to compute the aggregate of these 10 terms by simply summing over the hypotheses in the set, e.g.:

$$R(w) = \sum_s R_{s1}(w) \tag{6.3}$$

Where $R_{s1}(w)$ is the reference length of the 1-best hypothesis of s using weights w.

The key to Expected-BLEU optimization is that instead of only computing BLEU on the 1-best hypotheses, the algorithm computes the BLEU score on *every* hypothesis weighted by its posterior probability. The posterior probability for hypothesis sk is simply the normalized decoding probability:

$$P_{sk}(w) = \frac{e^{\gamma L_{sk}(w)}}{\sum_{j=1}^{N} e^{\gamma L_{sj}(w)}} \tag{6.4}$$

Since $L_{sk}(w)$ is a log probability, it is raised to the e power to compute the true probability, and then normalized over all of sentence s's hypotheses in the current n-best list. The free parameter γ controls the shape of the distribution, with a higher γ shifting more mass towards the 1-best hypothesis. In fact, as $\gamma \to \infty$, the probability of each 1-best hypothesis approaches 1.0, and thus Expected-BLEU tuning approaches 1-best tuning. However, we cannot simply set γ to a large value and simultaneously reap the benefits of both 1-best and Expected-BLEU tuning, as the optimization algorithm would become unreliable.

The Expected-BLEU objective function is the same[3] as Equation 6.2, except

[3]Other than a modified brevity penalty, which will be explained next.

the 10 terms are computed using the aggregate *expected value* rather than the aggregate 1-best. For example, instead of Equation 6.3, the reference length is defined as:

$$R(w) = \sum_{s=1}^{S} \sum_{k=1}^{N} P_{sk}(w) R_{sk} \tag{6.5}$$

It is important to note that the expected value of each of the BLEU terms is computed *independently*, which does not formally model the "expected-value of BLEU." The true expected value would be equivalent to randomly sampling a single hypothesis from each s proportional to its posterior probability, computing the aggregate BLEU over all S hypotheses, repeating the process a very large number of times, and taking the average. In this case the 10 terms in Equation 6.2 would be computed jointly for each hypothesis, rather than independently. However, I have found empirically that our method of Expected-BLEU tuning works very well even with this independent assumption.

The final caveat in our Expected-BLEU objective equation is to note that Equation 6.2 is not actually differentiable, due to the *brevity penalty*[4] formula:

$$B_{real}(x) = \min(1.0, e^{1-R/H}) \tag{6.6}$$

As a workaround, we[5] generated a differentiable formula in MATLAB that

[4]The brevity penalty is necessary because BLEU is a measure of precision on the hypothesis, and does not factor in recall or precision on the reference. Thus, if BLEU did not penalize short answers with the brevity penalty, it would prefer *very* short hypotheses.

[5]This step was performed by BBN researcher Spyros Matsoukas.

closely approximates $\min(1.0, x)$ in the range $0.9 < x < 1.1$. This formula is presented here *only* to make our Expected-BLEU formula easily replicable—it was simply generated using a least-squares fit and is not intrinsically meaningful. The estimated brevity penalty is given as $B(x)$ below:

$$S(x) = \frac{1}{1 - e^{x}} \tag{6.7}$$

$$\begin{aligned} B(x) = {} & 0.152 * S(-16.213x + 0.055) + \\ & 0.203 * S(21.421x + 1.111) + \\ & 0.271 * S(8.562x + 2.177) + \\ & -0.391 * S(-4.717x - 3.359) + 0.5297 \end{aligned} \tag{6.8}$$

The final Expected-BLEU formula is then:

$$ExpBLEU(w) = \left(\prod_{i=1}^{4} \frac{M^{(i)}(w)}{K^{(i)}(w)} \right)^{1/4} B(1 - R(w)/H(w)) \tag{6.9}$$

As mentioned earlier, Expected-BLEU is optimized using the gradient-based optimization algorithm L-BFGS. Thus, it is necessary to formulate the symbolic derivative of Equation 6.9 with respect to the decoding weights. The final equation for $\frac{dExpBLEU}{dw}$ is provided in the appendix, although the actual formulation of the derivative is *not* the original work of the author.

For both 1-best and Expected-BLEU optimization, the *shallowness* of the n-best list a major problem. In general, n-best lists are said to be shallow because they only represent a tiny fraction of the possible hypotheses that a decoder can produce

with a given rule file.[6] Often times, the weights that produce the optimal BLEU on the n-best list will produce very poor results when they are used to re-decode from scratch. Two major methods are used to mitigate this problem.

First, a long-standing method to handle the effect of n-best shallowness is *iterative optimization with n-best merging*. This means that the MT procedure is not to simply decode, optimize, and then decode again with the optimized weights to produce the final output. Instead, the procedure is to perform several iterations of decoding/optimization, and merge the n-best lists after each round. Generally 4-10 rounds of optimization are performed depending on the experimental conditions, and ideally the weights will have converged by the final round, meaning that additional rounds of optimization would not change them significantly.

I will briefly illustrate the purpose of n-best merging with a very common real-world example. Generally, BLEU optimization will produce MT output with a system-wide hypothesis-to-reference length ratio of about 100% (i.e., the hypotheses the same length as the references, overall). However, imagine that the initial weights are slightly mistuned, so the initial set of hypotheses has a length ratio of 105%. Due to n-best shallowness, the optimizer will likely reduce the *target word count* feature weight, which may produce output that is *too short* when the weights are used to re-decode, say a length ratio of 95%.

This itself is not necessarily a problem, because we perform several rounds of optimization. However, if n-best lists are *not* merged between the first and second

[6] If it is unclear, this is a general statement about n-best lists that applies to all types of decoders, as well as many other NLP applications.

rounds of decoding, the decoder may very will increase the *target word count* weight to produce 105% hypotheses again. The weights could then oscillate indefinitely, and would thus never converge. With n-best merging, in the second iteration the optimizer will be able to see that setting the *target word count* weight too high will bring low-BLEU hypotheses from the first iteration to the top, and will settle for a weight in the middle.

Second, I implemented a new procedure to restrict the amounts that the weights can change within a single iteration, called *weight regularization*. When using a weight regularization parameter, the Expected-BLEU procedure does optimize directly towards Equation 6.9, but instead optimizes towards the regularized term:

$$\text{argmax}_w \; ExpBLEU(w) - \alpha||w - w'||^2 \tag{6.10}$$

Where w' represents the initial weights for the current round of optimization, and α is a free parameter I generally set to 10^{-5}. Note that there is no term to restrict the global change of weights over many rounds of optimization. Additionally, the derivative of the regularization parameter is trivial to compute:

$$\frac{dExpBLEU}{dw} - 2\alpha(w - w') \tag{6.11}$$

The regularization parameter was not added because of the switch to Expected-BLEU tuning itself, since Expected-BLEU will have *greater* stability than Powell's

method when run under the same condition. Instead, it was added because when using discriminative features it is necessary to optimize *thousands* of weights rather than the normal 12. The use of a regularization parameter could theoretically cause the weights to converge at a sub-optimal value if α is set too high, but I have found it to make our optimization procedure more robust with no apparent side effects. Additionally, when regularization is used, the final weight magnitude of a binary-valued discriminative feature initialized at zero will be proportional to its effect on BLEU, which is very useful for feature selection. I will discuss this more in the next section.

6.2 Discriminative Features

Formally, a *discriminative* model directly estimates $P(X|Y)$, whereas a *generative* model estimates $P(X|Y)$ through $P(X,Y)$ and $P(Y)$ (or $P(Y|X)$ and $P(X)$). As an example, consider the lexical probability $P(t|s)$.[7] Our existing generative model counts the alignment links in the training data to produce $C(s,t)$ and $C(s)$, and then sets the probability as a maximum likelihood estimate: $P(t|s) = \frac{C(s,t)}{C(s)}$. A discriminative model, on the other hand, would choose the value of $P(t|s)$ which maximizes BLEU on a tuning set. In practice, it would be difficult to discriminatively train the values of $P(t|s)$ and $P(s|t)$ and use them in the existing lexical smoothing formulae for two reasons.

First, the lexical smoothing formulae are not linear, so a significantly different

[7] I do not want to shoehorn this into the formal definition by implying that $X = t$ and $Y = s$. The more reasonable formal definition would be to say we are estimating $P(X = s, t|Y = T, D)$, where T and D are the training and tuning data.

optimization paradigm would be needed. Second, it would be impractical to discriminatively train $P(t|s)$ for *every* $s \rightarrow t$, due to the relatively small size of the tuning set compared to the training. Instead, it is preferable to keep the existing generatively-trained lexical smoothing formulae, and add linear discriminative features for a *subset* of $s \rightarrow t$. In other words, the feature selection procedure chooses a set of source-target word pairs which are each assigned a binary feature that is "triggered" when that word pair is seen in a rule. A major advantage of discriminative training is that it is not necessary to actually come up with a generative story for $P(t|s)$. However, it is still necessary to perform *feature selection*, which is the procedure for choosing *which* $s \rightarrow t$ are going to be trained discriminatively.

The main rule of thumb for discriminative feature selection is that it is only possible to accurately estimate weights for discriminative features that are triggered a "fair number" of times in the tuning set. Obviously, if a feature is *never* triggered in the tuning set, its weight cannot be estimated at all. A weight *can* be estimated for any feature that is triggered 1 or more times, but if too many rarely-seen features are used, the tuning set will likely be severely *overfit*. A set of features is *overfit* if, by definition, performance on a tuning set is significantly better than performance on an unseen validation set. Generally, as the degrees of freedom (in this case, optimizable features) grow, the more problematic overfitting becomes [10]. A severely overfit model can cause significant problems—specifically, if a validation set happens to be *dissimilar* to the tuning set in some way, the results can be much *worse* on that set than they would be with a baseline model. Keeping this in mind, I will now describe the lexical translation feature mentioned above, as well as two more discriminative

lexical features which have been shown to be useful in practice.

6.3 Lexical Translation Features

The *lexical translation features* are the discriminative analogue to the lexical probabilities: Create binary features for some arbitrary set of word-pairs $s \rightarrow t$, and set the bin to 1.0 when s aligns to t in a rule, and 0.0 otherwise. All other word pairs that are not in the set trigger the "default" bin. Since discriminative training directly optimizes BLEU, it would seem that if it were possible to discriminatively estimate a weight for every possible $s \rightarrow t$, the results would have to be significantly better than using MLEs. The problem is that the MLEs are estimated on 10 million parallel training sentences, while our tuning set contains just 2000 sentences. Potentially, the size of the tuning set could be increased by several factors, but that still represents a small fraction of the full training data. If the size were increased beyond that, it would become completely impractical to decode and optimize on it as part of the normal development process.[8] Thus, I believe that it is only possible to accurately estimate weights for a small number of lexical translations, compared to the total number of lexical probability MLEs.

The feature selection process consists of creating bins for the n most common lexical translations $s \rightarrow t$ from the training, where s or t can be $NULL$. For all of my experiments, n is set to be 5000, creating 5001 total feature bins (including the "default"). Theoretically any increase in n should monotonically increase the

[8]Currently, decoding takes about 2.5 seconds per word averaged over the development set. Thus, decoding an entire 200 million word parallel training corpus would take about 16 years of CPU time.

BLEU score *on the tuning set*, but again, I believe that if a feature is not seen enough times, we will not be able to accurately estimate its weight for use on a validation set. In experiments where I increased n beyond 5000, the tuning set simply became more overfit with no benefit to the test sets. Now, I do *not* mean to imply that overfitting is a grave danger which must be avoided at all cost. Instead, I would just like to *mitigate* overfitting as much as is reasonably possible. In fact, all of the discriminative results presented in this chapter will be noticeably overfit—this is to some degree unavoidable when performing discriminative training.

I also include a secondary feature for added robustness called *lexical fertility*. The selection procedure is similar to the one above: For the n most common source words, create binary features for whether s aligns to 0, 1, 2, or 3+ words. In the experiments reported in this section, n is set to be 2500, resulting in $4 * 2500 + 1 = 10001$ total *lexical fertility* bins.

It should be noted that although the procedures above results in 15,000 feature bins, not every feature is triggered during the tuning process. Obviously, if a feature is never triggered, it receives a weight of 0 and thus has no effect. In Arabic-English, about 4200 out of 5000 direct translation bins are triggered, and 4300 out of 10,000 lexical fertility bins are triggered. Remember that this it *does* include many features which have been triggered only a handful of times and/or receive a weight near zero.

The results yielded by using these two types of discriminative features are presented in Table 6.2, under "Lex Trans - 1st Pass." Although there is a gain on the validation sets, it is clear that the results are significantly overfitted. For example, the largest gain on any validation set is 0.77 BLEU on Arabic Test 1, while the gain

for Arabic Tune is 2.98 BLEU. In order to reduce the effect of overfitting, I created a very simple process of *two-pass feature selection*: Take the final feature weight file from first-pass decoding (i.e., "Lex Trans - 1st Pass" in Table 6.2) and select the m feature bins with the highest magnitude weight from each discriminative feature type (i.e., *lexical translation features* and *lexical fertility*) as second-pass features. As an example, the top 10 highest-magnitude lexical translations for Arabic are presented in Table 6.1. After the second-pass features are selected, a second iteration of decoding is performed with the second-pass feature weights re-initialized to 0. Because weight regularization is performed, the feature bins with the highest magnitude of positive or negative weights presumably have the largest effect on the BLEU score. Although I make no claims of the optimality of this process, I have found it to work well without being impractically expensive.

Lexical Trans.	**Final Weight**
ywm → today	6.25
An → said	-5.95
Amrykyp → american	5.66
fY → in	-5.25
AwDH → explained	5.24
Ajl → order	5.07
lm → did	-4.92
Amrykyp → u.s.	-4.60
AEln → announced	4.50
dwlAr → dollars	4.48

Table 6.1: The 10 discriminative lexical translation features with the highest-magnitude decoding weights at the end of the first-pass decode.

In all discriminative training experiments, for all features[9], $m = 100$ for second

[9]In other experiments not presented here, we also use a number of *non*-lexical discriminative features that do not fit into the scope of this thesis.

pass feature selection. There are three reasons for this. First, "optimizing" the value of m is quite expensive, as it would require running a full decoding experiment for several different values of m and choosing the one that works best. Doing this for each discriminative feature type would require a large number of experiments, which would need to be rerun whenever there is a major change to the system. Second, m cannot be chosen based on the results of the *tuning* set, because the whole point is to reduce overfitting on the tuning set by lowering m. Thus, it would be necessary to effectively optimize m on one of the two test sets, which would no longer make that test set "clean." Third, although $m = 100$ is not optimal in all cases (e.g., Arabic Test 2 in Table 6.2), *overall* I have found that $m = 100$ significantly reduces overfitting while not harming the results on the test sets.

The final two pass results with $m = 100$ are presented in Table 6.2 under "Lex Trans - 2nd Pass." On Arabic the features produce significantly better results than the baseline, gaining 0.79 BLEU on Test 1 and 0.43 BLEU on Test 2. Overfitting is substantially reduced in the second pass, going from a 2.98 BLEU gain in the first pass to 1.05 BLEU. There is a smaller but consistent gain on Chinese.

In this case, we see that on the test sets, the results from the second-pass aren't any better than the results from the first-pass, although the overfitting was greatly reduced on the Tune set. The degree to which overfitting hurts an independent test set depends on how different the test set is from the tuning set, and in this case, the three sets are fairly similar. However, in general we believe that having a smaller number of free parameters (i.e., feature bins) is beneficial in that it reduces

	Chinese Newswire			Arabic Newswire		
	Tune	**Test 1**	**Test 2**	**Tune**	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*	*BLEU*	*BLEU*	*BLEU*
Baseline	33.96	25.50	34.35	56.80	53.11	55.60
Lex Trans - 1st Pass	**35.47**	25.84	34.46	**59.78**	53.88	**56.36**
Lex Trans - 2nd Pass	34.49	**25.90**	**34.60**	57.85	**53.90**	56.03

Table 6.2: Results for *lexical translations* and *lexical fertility*. The "1st Pass" uses all 5000 lexical translation and 10,000 lexical fertility features. The "2nd Pass" uses the *best* 100 lexical translation and top 100 lexical fertility features, taken from the first pass. Note that these baselines are different from the previous baselines because this work was done at a significantly later time, and the test sets had been modified to include new data.

the number of places where things can go wrong when decoding a new test set.[10] In the results presented here, two-pass training significantly reduces the number of free parameters (15,000 to 200) without harming the results on the development test sets, so we view we it as a helpful procedure, although perhaps not *essential*.

6.3.1 Part-of-Speech Translation Features

The coverage[11] of the discriminative features can be increased by using a part-of-speech based analogue to the lexical translation features, called *part-of-speech translation*. This feature estimates a weight for each part-of-speech pair $POS_{src} \rightarrow POS_{trg}$. For the target, all training data is POS tagged, and the rule extraction process is modified so that each rule includes a one-to-one mapping between target terminals and target POS tags. Whenever a rule is uniquely defined using more information, rule fragmentation occurs, which is generally harmful to a translation

[10]Where a "new test set" means a genuine unseen test set that may be unlike the training in some unpredictable way, not just development data.

[11]The "coverage" refers to the number of lexical translation pairs which trigger a non-default feature bin.

system. The solution to this problem is to perform *target tag merging*, so that if a set of rules differs only by POS tags, only the most frequent POS tag set will be kept. For example, take the two tagged rules with joint counts: $ciervo \rightarrow deer_{NN} = 8$, $ciervo \rightarrow deer_{NNS} = 3$.[12] In this case, they would be merged into the single rule $ciervo \rightarrow deer_{NN} = 11$.

For the source, the *test* data are tagged rather than the training. There could potentially be a POS mismatch between the sentence being decoded and the source side of the rule being applied, but since the source training is not tagged, there are no available statistics on how often this occurs. However, even if these mismatches do occur, it is preferable to use the POS tags of the test sentence.

In first-pass decoding, there is a bin for every possible $POS_{src} \times POS_{trg}$ tag pair, including $NULL$. This results in over 3000 feature bins in Chinese-English and over 10,000 in Arabic-English. However, most POS tag pairs never occur during tuning, so only about 1100 Chinese-English bins and 2100 Arabic-English bin are actually triggered in first-pass decoding. Second-pass decoding is performed in the manner described in the previous sub-section, again using $m = 100$. Results are presented in Table 6.3. Results are mixed, gaining 0.56 BLEU on Arabic Test 1, while losing 0.06 BLEU on Arabic Test 2.

[12]This example does not make total sense, because the Spanish plural for deer is "ciervos," not "ciervo." For this reason, POS tagging in bilingual translation rules is far less ambiguous than in monolingual text. However, target phrases with different POS tags still *do* occur.

	Chinese Newswire			Arabic Newswire		
	Tune	**Test 1**	**Test 2**	**Tune**	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*	*BLEU*	*BLEU*	*BLEU*
Baseline	33.96	**25.50**	34.35	56.80	53.11	**55.60**
POS Trans - 1st Pass	**34.76**	25.48	34.40	**58.64**	**53.72**	55.20
POS Trans - 2nd Pass	34.73	25.37	**34.58**	58.09	53.67	55.54

Table 6.3: Results for *part-of-speech translation*. "2nd Pass" feature selection again uses $m = 100$.

6.3.2 Corpus Balance Estimation Features

Our parallel training comes from a number of different corpus types, which can either be defined very by broad genres (e.g., "newswire" and "UN"), or more narrow collections (e.g.,"LDC2006E2" or "un-1995-jul-sep"). Since it is reasonable to believe that some corpora are "better"[13] than others, a weight w can be associated with each corpus c, so that each rule count and lexical count extracted from c is incremented by w instead of 1.0.[14] It is well established that when translating newswire data, for example, it is beneficial to weight newswire training data more than UN data. However, there is no single established method for estimating corpus weights. For a small number of broadly defined corpus types an ad-hoc weighting system such as binary search can be used. However, in order to estimate weights for a larger number of finely defined corpus types[15] our team had previously developed a gradient-descent based method that directly optimizes towards BLEU. The problem with this method is that due to its inherent computational complexity, it can only feasibly be run on a phrase based system and not the state-of-the-art hierarchical

[13]Not necessarily more fluent or accurate, but simply more similar to the test data.
[14]Note that this is exactly equivalent to duplicating the data w times, if w is an integer.
[15]There are 36 parallel training corpora in Arabic and 62 in Chinese.

system.

In order to overcome this shortfall in the existing corpus weight estimation procedure, I created a discriminative feature type called *corpus balance features*, which does not have the formal elegance of the standard procedure but *can* be optimized as part of the normal decoding model. Unlike the other discriminative features, this does not use binary bins, but instead defines an *accumulator bin* for each corpus. The procedure for calculating the feature scores is as follows:

1. Assume that each source sentence in the parallel training data is associated with a corpus c.

2. Accumulate the counts $Count(s, t, c)$ and $Count(s, t)$ over each alignment link in the training. This procedure is analogous to the accumulation of the lexical probability counts $Count(s, t)$, $Count(s)$, and $Count(t)$.

3. Calculate the maximum likelihood estimates of each translation conditioned on the corpus, i.e, $P(c|s, t) = \frac{Count(s,t,c)}{Count(s,t)}$. This probability table is loaded into the decoder.

4. In the decoder, create one feature bin for each corpus c. The feature score for corpus c in rule r is denoted as $F_r(c)$.

5. For each alignment link $s \rightarrow t$ in rule r, add the probability $P(c|s, t)$ to $F_r(c)$. In pseudocode:

   for all c **do**

   $F_r(c) = 0$

for all $s \rightarrow t$ in r **do**

$F_r(c) = F_r(c) + P(c|s,t)$

end for

end for

There are several things to note about this procedure. First, although $P(c|s,t)$ is technically a probability, it is inside of a summation rather than a product. So $F(c)$ is *not* analogous to the lexical probabilities, but instead represents the *percentage of times* that the lexical translations in a hypothesis came from corpus c.[16] This is the main intuition behind the feature: If a hypothesis contains lexical translations which come from "bad" (or "good") corpora an above-average amount of time, it will be discounted (or boosted) compared to alternate hypotheses. Because this feature represents a percent rather than a probability, the *real* values are used instead of the *log* values.[17] Additionally, since the corpora are of difference sizes, there is clearly going to be some prior expected distribution. However, normalization is not necessary because the features are linear and optimizable.

Second, this feature is not intrinsic to lexical translations, and can easily be performed on the rule level instead. The procedure in that case is to simply keep track of how often each rule r came from each corpus c and then set the feature value as $F_r(c) = P(c|r)$. I ran this experiment, and there was a gain on the tune set but a degradation on the test sets. Similar to my arguments in Chapter 3, this

[16] Also note that $P(c|s,t)$ is completely independent of the conditional probabilities $P(s|t)$ and $P(t|s)$, which are already accounted for in lexical smoothing.

[17] Since the decoding model is log-linear, all of the actual probabilities—rule translation, lexical smoothing, language model—are log-probs.

suggests that the feature is more reliable on the lexical level than the rule level because lexical translations are seen more often than rule translations. In other words, if a rule is seen only once, but it was seen in a "good" corpus, it will get a big boost from this feature, even though that boost is perhaps unwarranted.

Third, the "corpora" used for this feature do not need to be literal training collections, and can instead by *any* discrete assignment of the training sentences into bins. For example, imagine a case where each training sentence receives a "translation quality" score based on GIZA residuals. Instead of having to figure out an explicit model for using this score to weight the training, it would be possible to just discretize the scores into bins and use them with the *corpus balance estimation* feature. I have tried one experiment where the sentences are binned based on source:target word ratio, with the idea being that a sentence with an unusually high or low word ratio is likely to be a poor quality translation. This did not show any gain, but I may explore different "corpus" types in future work.

The results for corpus balance estimation are presented in Table 6.4. Note that two pass feature selection is *not* performed, as the feature space is already fairly small to begin with. Again, results are mixed, gaining 0.43 BLEU on Arabic Test 1 and 0.16 BLEU on Arabic Test 2, but nothing significant on Chinese.

	Chinese Newswire			**Arabic Newswire**		
	Tune	**Test 1**	**Test 2**	**Tune**	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*	*BLEU*	*BLEU*	*BLEU*
Baseline	33.96	25.50	34.35	56.80	53.11	55.60
Corpus Balance	**34.07**	**25.67**	**34.36**	**57.32**	**53.54**	**55.76**

Table 6.4: Results for *corpus balance estimation*. There are 36 features in Arabic and 62 in Chinese, equal to the number of parallel corpora.

6.3.3 Combined Discriminative Features

The results for all four discriminative feature types used jointly are presented in Table 6.5. Clearly lexical translations are the most powerful discriminative feature, but adding in POS translations and corpus balance estimation does help on Arabic. Overall, there is gain of 1.1 BLEU on Arabic Test 1 and 0.63 BLEU on Arabic Test 2. Overfitting is clearly worsened, but this may simply be a necessary evil when using discriminative features. On Chinese the results are more modest, gaining 0.4 BLEU and 0.25 BLEU on the test sets under using just the lexical translations. Even though Table 6.5 indicates that using corpus balance and POS translations on top of lexical translations is slightly harmful in Chinese, we do use all three features on both Chinese and Arabic in our standard system, since the features are inexpensive to compute and cumulative results have been positive.

In this chapter, I presented a novel optimization method called Expected-BLEU tuning, which is both robust and scalable to a very large number of decoding features. With this new optimization method in place, I implemented a procedure to *discriminatively train* the likelihood of particular lexical translations. This procedure simply creates binary feature bins for a set of $s \rightarrow t$ word pairs, where the feature score is set to 1.0 when the word pair is seen in a rule and 0.0 otherwise. This feature produces a gain of about 0.4-0.8 BLEU on Arabic and 0.3-0.4 BLEU on Chinese. I also described three other lexically-based, discriminative decoding features which produce a small gain on Arabic (but none on Chinese), bringing the total gain from the discriminative features 0.6-1.1 BLEU on Arabic.

	Chinese Newswire			Arabic Newswire		
	Tune	**Test 1**	**Test 2**	**Tune**	**Test 1**	**Test 2**
	BLEU	*BLEU*	*BLEU*	*BLEU*	*BLEU*	*BLEU*
Baseline	33.96	25.50	34.35	56.80	53.11	55.60
Lex Trans - 2nd Pass	34.49	**25.90**	**34.60**	57.85	53.90	56.03
POS Trans - 2nd Pass	34.73	25.37	34.58	58.09	53.67	55.54
Corpus Balance	34.07	25.67	34.36	57.32	53.54	55.76
All Lex Discrim Feats	**34.90**	25.79	34.50	**59.20**	**54.21**	**56.23**

Table 6.5: Results for combined *lexical translations*, *lexical fertility*, *part-of-speech translation*, and *corpus balance estimation*. All features except *corpus balance estimation* use two-pass feature selection with $m = 100$ as presented above.

Chapter 7

Conclusions

In this thesis, I presented a number of novel methods for either explicitly or implicitly modifying lexical (or word-to-word) translation probabilities. The first significant result is analytical: I demonstrated that the lexical smoothing function is significantly more important to our system than the rule probabilities, and posit that this is because the lexical probabilities are well-estimated while the rule probabilities are not.

Using the first result as a foundation, I then attempted to *bias* the lexical probabilities using both semantically similar documents from the training, and MT output from a first-pass decode, which is known as *self-adaptation*. The results show that semantic biasing does not produce a consistent gain, while the gain for self-adaptation is small but consistent.

Next, I showed that adding local context to the lexical probabilities can be beneficial, as long as the context-dependent lexical probabilities can be "backed-off" when they are believed to be unreliable. The best results are obtained by using only *previous word* and *next word* as independent context types, and interpolating the counts (not probabilities) using fixed weights. This method of *context-dependent lexical smoothing* produces a consistent gain of ∼0.5 BLEU on Chinese and Arabic.

Finally, I presented results for the discriminative training of lexical probabilities, as well as the discriminative training of two other lexically-based features. The features are said to be trained *discriminatively* because the feature values are directly optimized to maximize BLEU. In order to actually perform the training, I implemented a new optimization procedure called *Expected-BLEU optimization*, which can robustly tune thousands of features simultaneously. These features produce a gain of about ∼1.0 BLEU on Arabic, and ∼0.4 BLEU on Chinese.

7.1 Future Work

For context-dependent lexical smoothing, the most natural path to pursue is the use of additional context models. As a specific example, in Arabic, I have found that training an undiacratized[1] system and then using the diacritics as an additional CLS model is superior to a number of alternate techniques, such as training a fully diacritized system, using a fixed threshold to decide whether a word should be diacritized, and using a decision tree.

[1]In the Arabic writing system, words can be written with or without diacritics, which are vowel markers that can disambiguate meaning. Almost everything is written *without* diacritics, but they can be generated automatically with high accuracy using the Sakhr morphological analyzer.

Another possible area to explore is long-distance context, which can be used in the forward direction since the entire source sentence is known at decode time. For example, I have tried extracting source-source word pairs with high sentence level co-occurrence from the training, and then using the context model $P(t|s_i, s_j)$, where s_j can occur *anywhere* else in the sentence. I have also used the full parse tree instead of just the POS tags, for example using the model $P(t|s, VP)$, which is triggered if VP is an ancestor constituent of s.

Additionally I am experimenting with unsupervised clustering of words into context classes. In other words, if the source words s_i and s_j tend to change the meaning of the words they precede in similar ways, then they would be put into the same class C_k. The context model would thus be $P(t|s, C_k)$, where C_k is the class of the previous word.

For the discriminative training, I am not just focusing on new feature types, but also better estimation of the existing features. The simplest way to do this would be to increase the size of the tuning set. Attempts to automatically select data from the training to be used in the tuning set have not been successful, likely because the training data is noisy and the translations are often inexact. In future work, I will experiment with a new, high-quality tuning data set (20,000 segments); this should answer any questions about tuning set size, outside of the computational costs. There may also be better methods of reducing overfitting in the current tuning set, although so far I have not found anything that works better than our simple two-pass feature selection procedure.

Finally, the idea of semantic biasing, as described in Section 4.1, is a concept

with a strong intuitive foundation. The system is trained on several million documents, all with equal weights.[2] It would seem to follow, then, that when translating a document about, say, NASA, then selecting 25 documents about NASA from the training and giving them a higher would be beneficial. However, I have tried a number of different methods to do this (only one of which is described in this thesis), and none have produced a non-trivial gain. Even so, I still believe that some method of semantic biasing does have the potential to produce substantial gains.

[2]This is not technically true due to our corpus weighting procedure, but corpus weighting is very broad compared to what we're talking about here.

Appendix A

Expected-BLEU Derivative

In order to perform gradient-based optimization, we must compute the the derivative of the Expected-BLEU formula, Equation 6.9, with respect to the decoding weights w. This differentiation was performed by a BBN researcher Spyros Matsoukas.

Let:

$$ExpBLEU(w) = e^{Q(w)}V(w)$$

Where:

$$\begin{aligned}
Q(w) &= \frac{1}{4}\left(\sum_{i=1}^{4}\ln(\sum_s\sum_k P_{sk}(w)M_{sk}^{(i)}) - \ln(\sum_s\sum_k P_{sk}(w)K_{sk}^{(i)})\right) \\
V(w) &= B(1-\frac{\sum_s\sum_k P_{sk}(w)R_{sk}}{\sum_s\sum_k P_{sk}(w)H_{sk}}) \\
u &= \frac{\sum_s\sum_k P_{sk}(w)R_{sk}}{\sum_s\sum_k P_{sk}(w)H_{sk}} \\
V(w) &= B(1-u)
\end{aligned}$$

and B(x) is defined as in Equation 6.8.

Then the derivative is:

$$\begin{aligned}
\frac{dExpBLEU}{dw} &= \sum_s \sum_k \frac{dExpBLEU}{d\ln(P_{sk})} * \frac{d\ln(P_{sk})}{dw} \\
&= \sum_s \sum_k \sum_j \frac{dExpBLEU}{d\ln(P_{sk})} * \frac{d\ln(P_{sk})}{d\ln(L_{sj})} * \frac{d\ln(L_{sj})}{dw} \\
&= \sum_s \sum_k \sum_j \frac{dExpBLEU}{d\ln(P_{sk})} * \frac{d\ln(P_{sk})}{d\ln(L_{sj})} * F_{sk}
\end{aligned}$$

$$\ln(P_{sk}) = L_{sk}(w) - \ln(\sum_j e^{L_{sj}(w)})$$

$$\begin{aligned}
\frac{d\ln(P_{sk})}{d\ln(L_{sj})} &= \delta_{kj} - \frac{e^{L_{sj}(w)}}{\sum_m e^{L_{sm}(w)}} \\
&= \delta_{kj} - P_{il}(w)
\end{aligned}$$

where:

$$\delta_{kj} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{if } k \neq j \end{cases}$$

$$\begin{aligned}
\frac{dExpBLEU}{d\ln(P_{sk})} &= e^Q \frac{dV}{d\ln(P_{sk})} + e^Q \frac{dQ}{d\ln(P_{sk})} * V \\
&= e^Q (\frac{dV}{d\ln(P_{sk})} + V \frac{dQ}{d\ln(P_{sk})})
\end{aligned}$$

$$\begin{aligned}\frac{dV}{d\ln(P_{sk})} &= \frac{dB}{d(1-u)}\frac{d(1-u)}{d\ln(P_(sk)} \\ &= \Delta B(1-u)*(-1)*\frac{du}{d\ln(P_{sk})}\end{aligned}$$

$$\begin{aligned}\frac{du}{d\ln(P_{sk})} &= \frac{e^{\ln(\sum_s\sum_k P_{sk}(w)R_{sk})-\ln(\sum_s\sum_k P_{sk}(w)H_{sk})}}{d\ln P_{sk}} \\ &= u\left(\frac{P_{sk}(w)R_{sk}}{\sum_s\sum_k P_{sk}(w)R_{sk}} - \frac{P_{sk}(w)H_{sk}}{\sum_s\sum_k P_{sk}(w)H_{sk}}\right)\end{aligned}$$

$$\frac{dQ}{d\ln(P_{sk})} = \frac{1}{4}\sum_{i=1}^{4}\left(\frac{P_{sk}(w)M_{sk}^{(i)}}{\sum_s\sum_k P_{sk}(w)M_{sk}^{(i)}} - \frac{P_{sk}(w)K_{sk}^{(i)}}{\sum_s\sum_k P_{sk}(w)K_{sk}^{(i)}}\right)$$

Bibliography

[1] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[2] Philip Blunsom, Trevor Cohn, and Miles Osborne. A discriminative latent variable model for statistical machine translation. In *Proceedings of ACL 2008*. Association for Computational Linguistics, 2008.

[3] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311, 1993.

[4] Clara Cabezas and Philip Resnik. Using wsd techniques for lexical selection in statistical machine translation. In *Technical report, Institute for Advanced Computer Studies, University of Maryland.*, pages 61–72, 2005.

[5] Marine Carpuat and Dekai Wu. Improving statistical machine translation using word sense disambiguation. In *In The 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007*, pages 61–72, 2007.

[6] Boxing Chen, Min Zhang, Aiti Aw, and Haizhou Li. Exploiting n-best hypotheses for smt self-enhancement. In *HLT '08: Proceedings of the 46th Annual*

Meeting of the Association for Computational Linguistics on Human Language Technologies, pages 157–160, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[7] David Chiang. Hierarchical phrase-based translation. *Comput. Linguist.*, 33 (2):201–228, 2007. ISSN 0891-2017.

[8] David Chiang, Adam Lopez, Nitin Madnani, Christof Monz, Philip Resnik, and Michael Subotin. The hiero machine translation system: Extensions, evaluation, and analysis. In *In Proceedings of HLT/EMNLP 2005*, pages 779–786, 2005.

[9] David Chiang, Kevin Knight, and Wei Wang. 11,001 new features for statistical machine translation. In *NAACL '09: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 218–226, Morristown, NJ, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-41-1.

[10] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, 1995. ISSN 0360-0300.

[11] Marcello Federico and Nicola Bertoldi. How many bits are needed to store probabilities for phrase-based translation? In *In Proc. of NAACL Workshop on Statistical Machine Translation*, pages 94–101, 2006.

[12] George Foster, Roland Kuhn, and Howard Johnson. Phrasetable smoothing

for statistical machine translation. In *EMNLP '06: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 53–61, Morristown, NJ, USA, 2006. Association for Computational Linguistics. ISBN 1-932432-73-6.

[13] Philipp Koehn Franz, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 127–133, 2003.

[14] K. Gimpel and N. Smith. Rich source-side context for statistical machine translation. In *ACL-08: HLT third workshop on statistical machine translation*, pages 9–17, 2008.

[15] Zhongjun He, Qun Liu, and Shouxun Lin. Improving statistical machine translation using lexicalized rule selection. In *COLING '08: Proceedings of the 22nd International Conference on Computational Linguistics*, pages 321–328, Morristown, NJ, USA, 2008. Association for Computational Linguistics. ISBN 978-1-905593-44-6.

[16] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, pages 181–184 vol.1, 1995.

[17] Philipp Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *AMTA*, pages 115–124, 2004.

[18] Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 761–768, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[19] Dong C. Liu, Jorge Nocedal, and Dong C. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[20] Franz J. Och and Hermann Ney. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417+, 2004.

[21] Franz J. Och, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28, University of Maryland, College Park, MD, June 1999.

[22] Franz Josef Och. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 160–167, Morristown, NJ, USA, 2003. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1075096.1075117.

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

[24] Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alexander Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine

Eng, Viren Jain, Zhen Jin, and Dragomir R. Radev. A smorgasbord of features for statistical machine translation. In *HLT-NAACL*, pages 161–168, 2004.

[25] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[26] Adam Pauls, John Denero, and Dan Klein. Consensus training for consensus decoding in machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1418–1427, Singapore, August 2009. Association for Computational Linguistics.

[27] D. Povey and P.C. Woodland. Minimum phone error and i-smoothing for improved discriminative training. In *Proc. IEEE Internat. Conf. Acoustics, Speech, Signal Processing*, pages 105–108, 2002.

[28] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2): 155–162, February 1964.

[29] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings AMTA*, pages 223–231, August 2006.

[30] Matthew Snover, Bonnie Dorr, and Richard Schwartz. Language and transla-

tion model adaptation using comparable corpora. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 857–866, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[31] Nicolas Stroppa, Antal van den Bosch, and Andy Way. Exploiting Source Similarity for SMT using Context-Informed Features. In *Proceedings of the 11th Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 231–240, 2007.

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

[33] Roy W. Tromble, Shankar Kumar, Franz Och, and Wolfgang Macherey. Lattice minimum bayes-risk decoding for statistical machine translation. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 620–629, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[34] Nicola Ueffing. Self-training for machine translation. In *NIPS workshop on Machine Learning for Multilingual Information Access*, 2006.

[35] Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word

alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics*, pages 836–841, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[36] I.T. Witten and T.C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. In *IEEE Transactions on Information Theory*, volume 37(4), pages 1085–1094, 1991.

[37] Jinxi Xu, Ralph M. Weischedel, and Chanh Nguyen. Evaluating a probabilistic model for cross-lingual information retrieval. In *SIGIR*, 2001.

[38] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 523–530, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[39] R. Zens and H. Ney. Improvements in phrase-based statistical machine translation, 2004.