Parameterizing Phrase Based Statistical Machine Translation Models: An Analytic Study

by

Daniel Cer

B.S., University of Colorado, 2002

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2011

This thesis entitled:
Parameterizing Phrase Based Statistical Machine Translation Models: An Analytic Study
written by Daniel Cer
has been approved for the Department of Computer Science

_____
James H. Martin

_____
Prof. Daniel Jurafsky

_____
Prof. Christopher D. Manning

_____
Prof. Wayne Ward

_____
Prof. Richard Byrd

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Cer, Daniel (Ph.D., Computer Science)

Parameterizing Phrase Based Statistical Machine Translation Models: An Analytic Study

Thesis directed by Prof. James H. Martin

The goal of this dissertation is to determine the best way to train a statistical machine translation system. I first develop a state-of-the-art machine translation system called Phrasal and then use it to examine a wide variety of potential learning algorithms and optimization criteria and arrive at two very surprising results. First, despite the strong intuitive appeal of more recent evaluation metrics, training to these metrics is no better than the older traditional approach of training to BLEU. Second, the most widely used learning algorithm for training machine translation systems, called minimum error rate training (MERT), works no better than standard machine learning algorithms such as log-linear models. This result demonstrates that machine translation does not require using a special purpose learning algorithm, but rather can be approached in a manner similar to other natural language processing and machine learning tasks. These results have a number of important implications. Contrary to existing beliefs, work on improving machine translation evaluation metrics and then training to the improved metrics will not in itself result in improved translation systems. Even more significantly, the widespread usage of MERT has limited the sort of models that can be used for machine translation, as it does not scale well to large numbers of features. If it is not necessary to use MERT to train competitive systems, machine translation can be treated similarly to any other natural language processing task with models that include arbitrarily large feature sets.

Dedication

To my family.

Acknowledgements

I would like to thank Dan Jurafsky and Chris Manning for the opportunity to work with the Stanford natural language processing group, as well as all the present and former members of the Stanford machine translation group and my other collaborators, particularly Huihsin Tseng, Jason Brenier, Marie-Catherine de Marneffe, Michel Galley, Pi-Chuan Chang, Sebastian Padó, Spence Green, Bryan Pellom, and Randall C. O'Reilly. I would also like to thank the members of my committee for their valuable feedback.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

In this dissertation, I investigate the effectiveness of different approaches to training statistical machine translation models. I present experimental results on the use of different optimization algorithms as well as the benefits, disadvantages and trade-offs involved in using different training criteria. A novel approach to training models to slower evaluation metrics is also presented. The experimental results challenge some of the long held beliefs regarding machine translation model training, with strong implications for ongoing research into new training criteria.

Training decoding models for statistical machine translation differs from most other machine learning and natural language processing tasks in that there is a very close relationship between the methods used to evaluate system performance and model learning. Most machine learning tasks are approached by first formulating a model that can map from a set of inputs to the desired output representation. The parameters of the model are then trained to make the desired mapping using some sensible objective function that trades off the accuracy of the mapping over some set of training data points and how well the solution is expected to generalize. For any given set of model parameters, the objective function assigns a single numerical score that reflects how well the model fits the training data combined with some regularization term that prefers simpler more general models. Learning then reduces to searching for the model parameters that achieve the best possible score according to the objective function.

After training is completed, the quality of the resulting model is evaluated by scoring its performance on held-out data that was excluded from training. The calculation of this score is problem specific. Problems involving assigning appropriate category labels to each data point will often just be scored according to the

label accuracy, $\left(\frac{\text{correct labels}}{\text{total labels}}\right)$. Problems where accuracy on rare labels is important can be evaluated using slightly more sophisticated metrics such as the precision, recall, and F-score of some class of interest. For most tasks, choice of evaluation metric is reasonably straightforward and is independent of the objective function used to train the model. The objective function used during training only depends on the choice of modeling frameworks, and task specific evaluation criteria allows the performance of different modeling frameworks to be compared to each other.

Training machine translation models directly combines evaluation metric selection with selection of the criterion optimized during training. Whatever criterion will be used to ultimately evaluate the model is typically also used as the objective function that is optimized during model training. The reason machine translation systems are trained by directly fitting evaluation metrics is due to two factors. The first is that machine translation differs from most other machine learning and natural language tasks in that there are typically numerous acceptable translations of any non-trivial source sentence. Due to the structure of the machine translation system, some of these might be effectively easier for it to produce than others. Other acceptable translations might be out of reach to the system due to a vocabulary or idiom mapping between the two languages that the system does not know about or the system's limited capacity for reordering material during translation. The second reason machine translation decoding models are trained by directly fitting evaluation metrics is that early experiments suggested that this approach performed better than using more traditional machine learning methods (Och, 2003). However, results presented in subsequent work suggest that more mainstream learning algorithms may be competitive with exact metric fitting. Chiang et al. (2008) demonstrates that model tuning using the MIRA algorithm performs as well as exact metric fitting. But it is not clear whether this result is idiosyncratic to MIRA and under what conditions other popular machine learning algorithms can be successfully used for training machine translation models.

There are, however, unique problems that arise when typical machine learning algorithms are used to train machine translation systems. Supervised machine learning tasks involve training a model to prefer certain correct output targets for whatever input the model is provided. For machine translation, we have correct reference translations that could be used as targets during learning. However, the existence of numerous acceptable translations and limitations in what translations the model can produce creates a situation where

it is not really clear what should be used as the target of learning. Using the reference translations as the learning targets can result in the loss of over half the sentences in the training data, because the system is just not capable of producing the reference translation. For the reference translations that it can produce, some of them are only reachable by pathological derivations such as incorrectly translating punctuation marks or functions works into content words and meaningful phrases (Liang et al., 2006a).

Training to an evaluation metric allows the system to produce the closest translation it can to the reference given its limitations. It also results in the system using more sensible derivations, since the derivations it uses are selected based on how well they work for all of the sentences in the training data in which they might be used. Such training then bridges the gap between what is in the reference translation and what is possible given the system being trained. It also plausibly allows improvements in evaluation metrics to improve training and the quality of the resulting translations. New evaluation metrics that broaden the set of high scoring correct and mostly correct sentences, while still limiting the number of incorrect translations assigned a high score, give the system more flexibility in the good translations it can try to produce. This in turn increases the odds of it being able to learn to produce one of the better translations.

Using the system's performance on an evaluation metric as the criterion that is maximized during training brings with it a number of issues and challenges. The metrics used to evaluate translation systems, such as BLEU, TER, METEOR, mWER, only examine the translations produced by the system. Small changes in the decoding model parameters will usually not change these translations. Making small changes then typically does not alter the score assigned to the system by the evaluation metric. However, at certain critical points, changes in the parameter values will result in a different translation candidate receiving the highest relative score according to the decoding model. This new translation candidate will then be supplied to the evaluation metric, altering the evaluation score assigned to the system.

The resulting training objective being optimized can be characterized as being full of flat plateaus with discontinuous jumps from one plateau to another. Such objective functions are more difficult to work with since they cannot be approached by gradient descent based methods. A number of approaches are taken within the machine translation community to optimize such a space including downhill simplex and Powell's method. The popular CMERT software package just uses coordinate descent with an efficient derivative-

free line search. There is little work on systematically evaluating which of these approaches works best for machine translation.

Training to evaluation metrics also brings with it the question of what evaluation metric should be used. Training to a specific metric should produce a system that does particularly well on that metric. However, for this improvement to be meaningful, it must also produce translations that are actually objectively better. It is possible to imagine training to a metric and getting a system that does a good job of maximizing whatever is being measured by the metric, but that does so in a way that either does not improve or perhaps even harms the actual translation quality.

For example, imagine there was a metric that simply measured how many words were present in translations produced by the system that were also present in reference translations created by human translators. If the system produced "The election held" and the correct translation was "The election was held on Tuesday", it would be rewarded with three points for matching three words in the reference. While such a metric could be used to estimate the quality of the system's translations, using it to train the decoding model would likely result in a system that produces excessively long translations. The system would try to produce as many words as possible, since the more words it produces the greater the odds that one or more of them will match the words in the reference translation.

More sophisticated metrics may be harder to cheat, but training will try to exploit any and all properties of the metric. Training to evaluation metrics makes it essential we verify that improvements in the score achieved by maximizing a metric actually reflect true improvements in translation quality. Unfortunately, most published work on machine translation has no or little human evaluation of the translations produced by the system. Introducing a new feature to a system with a corresponding new parameter that can be set during training may just allow the system to better fit the preferences of the evaluation metric used for training without actually improving translation quality. There has been a substantial amount of work that has examines how well the scores returned by various machine translation metrics correlate with human judgments. However, there has been no work that examines how training to different evaluation metrics affects the true quality of machine translations as evaluated by human judges.

Popular evaluation metrics largely examine the number and length of overlapping words that are

present in both a machine translation and one or more reference sentences. They differ then largely in how they go about measuring that overlap and how much variation they allow in what they consider a match. Some metrics such as the popular BLEU score (Papineni et al., 2002) only allow exact matches between words in the reference translation and the words in a translation being scored. Others such as TERp (Snover et al., 2006) and METEOR (Lavie & Denkowski, 2009) allow for matches of words with similar meanings, or even paraphrases of various multi-word expressions.

Since they perform such a superficial analysis, these metrics are fairly fast. This is a critical property for metrics that will be used to train a system, since during training hundreds of thousands of evaluations maybe performed. However, it is possible that such a superficial analysis handicaps training. A machine translation system may be able to produce a translation that closely matches the meaning of a human reference translation, but that is worded quite differently than the available references. A deeper analysis of the meaning of both sentences could reveal their similarities. But, if the wording is different enough, a simple word sequence overlap based metric will incorrectly give the translation a very low score. If there is a choice between producing the differently worded but correct translation or an incorrect translation with more overlapping word sequences, training to a simple overlap based metric will cause the system to prefer the incorrect translation.

Metrics that perform a deeper analysis of the translations being evaluated do exist. For instance, the Stanford Entailment metric scores sentences according to whether the information in the reference translation seems to entail what is in the machine translation and vice versa. However, this metric is substantially slower than the simpler word sequence overlap approaches, taking seconds to score each pair of sentences that it is given. Since a very large number of evaluations need to be performed during training, slow metrics that perform deep analysis have not been used for training machine translation systems. This is unfortunate since such metrics could plausibly improve the translation quality in ways that are not possible with simpler more superficial metrics. No work exists that has attempted to speed up or adapt such a metric for the purposes of training.

This work investigates the issues involved with using machine translation evaluation metrics to parameterize phrase-based statistical machine translation models as well as the use of alternative learning methods.

It investigates different methods of searching the parameter space, the effects of using different popular evaluation metrics, as well as the use of the new Stanford entailment based evaluation metric that performs a much deeper analysis of meaning than the simple surface analysis performed by most other metrics. Finally, I examine the use of popular algorithms from the mainstream machine learning community for training machine translation models. The remainder of this chapter is organized as follows. The next section provides a brief overview of phrase-based machine translation. After that, motivation for the work presented here is provided. The chapter closes with a summary of prior work and an overview of the organization of the rest of the material presented within this document.

1.1 Overview of Phrase-Based Machine Translation

Phrase-based machine translation systems translate a sentence *f* into translation *e* by segmenting *f* into a series of phrases that are translated and rearranged to construct *e*. The sentence being translated is known as the **source** , while the translation being constructed is called the **target**. As seen in figure 1.1, translations are produced from left-to-right, with the system selecting at each time step one or more contiguous words in the original sentence and then appending the translation of the selected material to the end of the current partial translation.

The phrases used to construct the translation are selected from a phrase-table. Phrase-tables can contain numerous entries for the phrases being translated in the source. While some entries may capture different meanings of the source words being translated, much of the variation comes from different possible grammatical variations. As seen in table 1.1, an English to French phrase-table allows for the French phrase "le budget de" to be translated as "the budget" followed by a number of different function words such as "by", "from", "of", and "on". It also allows the translation to be prefixed by "," "to", or even "'s". These variations allow the decoder to select whatever grammatical glue is most appropriate given the reset of the material in the sentence being translated. It also gives the decoder stylistic flexibility in being able to render the translations as either "the budget of X" or "X's budget".

Since the final translated sentence is built-up one phrase at a time, constructing a good translation requires the system to always select the correct material to translate and then choose a translation of that

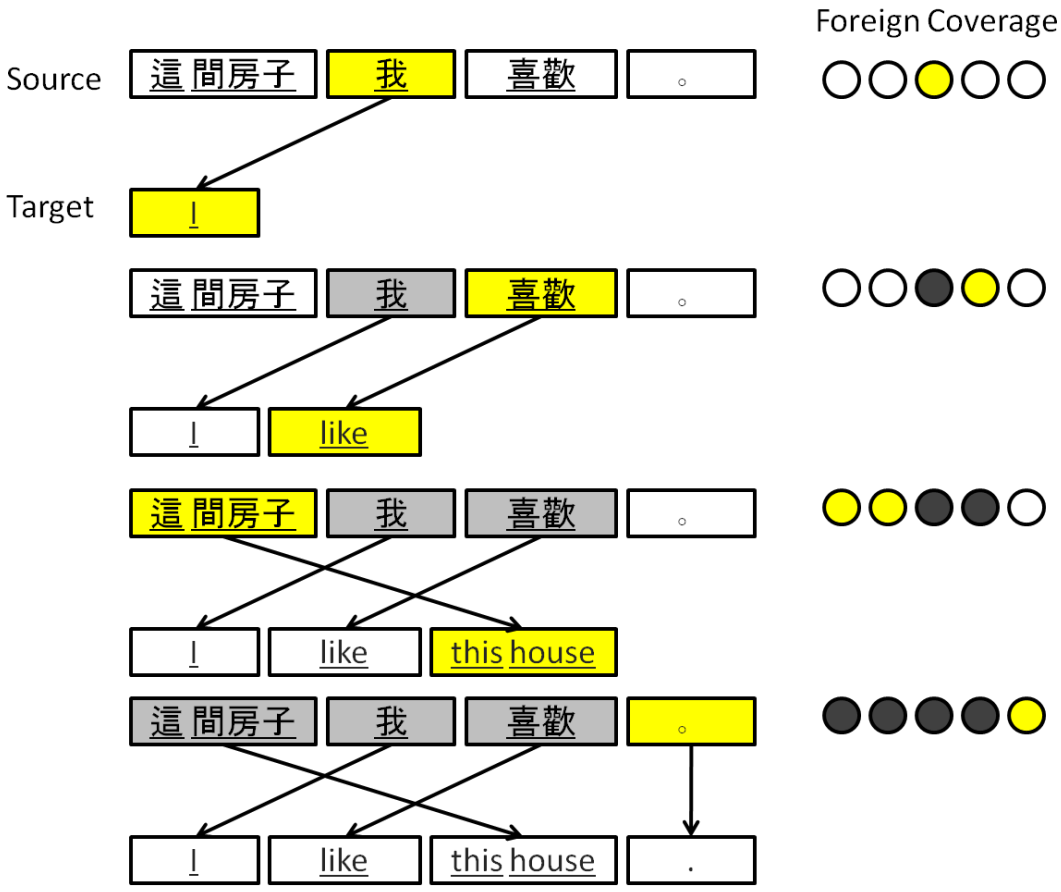

Figure 1.1: **Phrase-Based Decoding**: The translation is produced from left-to-right by selecting and then translating words and phrases in the original sentence. In addition to the current partial translation, the decoder keeps track of what material has been translated so far as well as the location of the last foreign material that was translated.

material that not only preserves the original meaning but that also is grammatically correct given the existing partial translation. This process is guided by a decoding model that consists of a weighted combination of knowledge sources. Known as decoding model **features**, these knowledge sources can be partitioned into three classes. The first are the translation model features, which score the probability that a phrase in isolation is a good translation of a corresponding phrase in the source material. The second are language modeling features. These features score the fluency and grammaticality of the translation being constructed. This is typically done using *n*-gram language models, but there has been some work in using syntactic dependency language models (Galley & Manning, 2009). Finally, the third class of features consists of re-ordering models that attempt to capture word order differences between the source and target languages.

Source Phrase	**Target Phrase**	P_{lex}(**e** |**f**)	P_{lex}(**f** |**e**)	P(**e** |**f**)	P(**f** |**e**)	Phrase Penalty
le budget de la	the budget for the	0.018	0.006	0.143	0.013	2.718
le budget de la	the budget of the	0.135	0.014	0.714	0.079	2.718
le budget de la	to the budget of the	0.333	0.014	0.143	0.079	2.718
le budget de	's budget ,	0.25	0.017	0.010	0.022	2.718
le budget de	's budget for	1	0.010	0.010	0.001	2.718
le budget de	's budget	0.248	0.017	0.377	0.022	2.718
le budget de	, the budget for	0.125	0.023	0.010	0.022	2.718
le budget de	in its budget	0.1	0.022	0.010	0.002	2.718
le budget de	its budget	0.023	0.022	0.010	0.002	2.718
le budget de	tax	0.001	2.8e-6	0.010	1.2e-4	2.718
le budget de	the budget by	0.2	0.023	0.010	0.010	2.718
le budget de	the budget for	0.284	0.023	0.276	0.022	2.718
le budget de	the budget from	0.5	0.037	0.010	0.011	2.718
le budget de	the budget of	0.767	0.053	0.235	0.138	2.718
le budget de	the budget on	0.25	0.014	0.010	0.012	2.718
le budget de	this whole budget of	1	0.011	0.010	0.005	2.718
le budget de	to the budget of	0.333	0.053	0.010	0.138	2.718

Table 1.1: **Phrase Table Entries:** Entries in a French to English phrase-table for *le budget de* and *le budget de la*. The phrase-table contains both lexicalized and unlexicalized phrase translation probabilities going from both French to English and English to French. The last column is a phrase penalty.

System	Features	Description
Pharaoh	Language model	P(e), n-gram probability of sentence e.
	Translation model	P($e|f$), P($f|e$), $P_{lex}(e|f)$, $P_{lex}(f|e)$, probability of phrases in e being translated as phrases in f.
	Phrase penalty	Count of phrases used to produce e from f.
	Word penalty	Count of words in e.
	Distortion penalty	Measure of linear distance between phrases in e and their counter parts in f.
	Unknown word	Number of untranslated words from f in e.
Moses	Lexical reordering	$P_m(f_1f_2|f_1)$, $P_s(f_1f_2|f_1)$,$P_d(f_1f_2|f_1)$, $P_m(f_1f_2|f_2)$, $P_s(f_1f_2|f_2)$. $P_d(f_1f_2|f_2)$, Probability of adjacent phrases in f being either monotone (m), swapped (s), and non-adjacent/discontinuous (d) in e.

Table 1.2: **Machine Translation Feature Sets:** Common decoding model features, grouped by the translation software packages that led to their widespread usage.

Table 1.2 illustrates the features commonly used in modern phrase-based machine translation systems to guide the selection and translation of material from the original sentence. Older translation systems based on the Pharaoh software package have a total of nine features: a language model score, 4 translation model scores, a phrase count, a target word count, a distortion penalty, and an unknown word penalty. Model weights are learned for only eight of these features, as the unknown word feature is assumed to have a weight of 1. The Moses software package adds support for six additional features that model the reordering of phrases in e from their relative locations in f. The Phrasal decoder used for my experiments supports the Pharaoh features, the additional lexical reordering features introduced by Moses, and a hierarchical extension of the Moses reordering features.

1.1.1 Learning Model Weights

Recall that within other applications of natural language processing and machine learning, models are typically trained to map the input provided to the system to certain static targets. However, within machine translation, there are typically numerous equally valid translations of any non-trivial sentence. Given the structure of a model, certain correct translations might be easy to produce, while producing others would require questionable translations of some of the content in the source sentence. Moreover, many models may not even be able to generate some of the correct translations.

The training technique known as minimum error rate training (MERT) avoids target selection issues by tuning model parameters to directly maximize the score achieved on some automated measure of translation quality (Och, 2003). The method works by making the learning objective identical with the score produced by the selected evaluation metric. This learning objective is piecewise constant, since the evaluation metric only examines the highest scoring translations produced by the system and these translations will typically not change for most small perturbations of the model weights. However, at certain critical points, the score assigned to another translation will exceed the score of the previous best translation. At these points a new translation is provided to the evaluation metric and the metric will likely change the score it assigns to the system.

Producing machine translations for any given set of model weights is a relatively slow process. Having

a decoder translate a standard dataset used for model training can take up to an hour or more. This makes it impractical to run the decoder for every point in parameter space that will be explored during training. To make training tractable, learning is performed by first running the decoder and having it generate *n* high scoring candidate translations for each sentence in the training data using manually set weights. These *n*-best lists of candidate translations are then used by MERT to search for a new set of weights that achieves a higher evaluation metric score when they are used to *select translation hypotheses just from the n-best lists*. In this way, the *n*-best lists allow MERT to quickly simulate decoding using a large number of different model parameters.

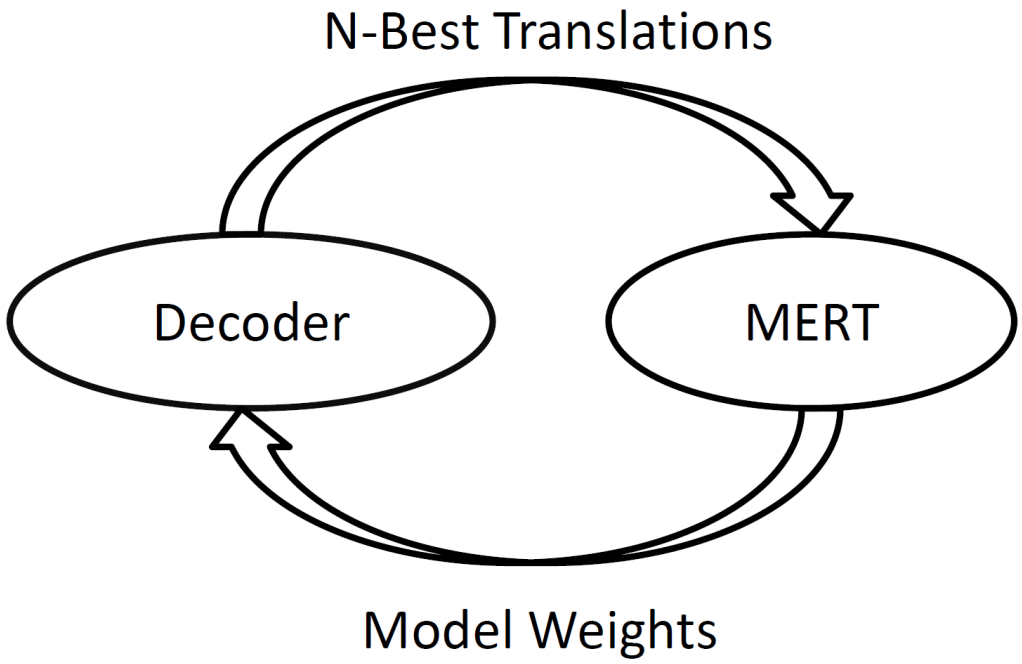

Figure 1.2: **MERT Training Loop:** MERT alternates between the decoder producing new *n*-best lists of translations and using those *n*-best lists to learn new model weights. Learning is done in batches, with the decoder translating all of the sentences in the training data before the model weights are updated by running the learning algorithm over the new *n*-best lists.

The *n*-best lists are, however, at best only a rough approximation of what the decoder would do when given various different parameter values. As shown in figure 1.2, this approximation can be made iteratively more accurate by using the parameter values returned by learning over the current *n*-best lists to re-run the decoder to obtain additional *n*-best list entries that can be then combined with those found during earlier

decoder runs. This process continues until either the decoder returns only *n*-best list entries that were discovered during prior iterations or the parameter values learned from one iteration to the next differ by at most some very small value ε.

1.1.2 Progression of MERT

Say that we have a simplified decoder with three features: a language model score, a translation model score, and linear distortion. The language model score is the log probability assigned by an *n*-gram language model to the translation. The translation model score is the sum of the log probabilities assigned to the translation of individual phrases. Linear distortion is a measure of the linear distance between the position in the original sentence of the first word in a phrase being translated and the position in the original sentence of the end of the phrase that was just translated.

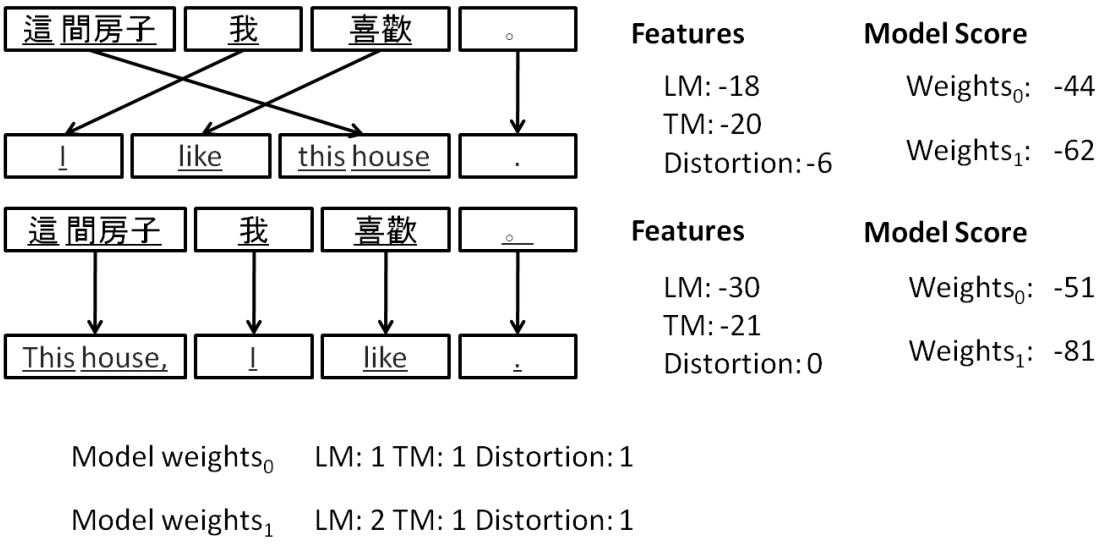

Figure 1.3: **Selecting Translations Using Model Weights:** The decoder chooses the appropriate translation of a foreign sentence according to what translation receives the highest score from the decoding model. Learning then consists of adjusting the model weights so that the translations preferred by the decoder are as close as possible to reference sentences produced by human translators.

Say that, during the first iteration of training, the weights for all of the features are set to 1.0. This will cause the decoder to equally weight the model scores for the isolated translation of individual phrases, the reordering of phrases when going from the original sentence to the translation, and the *n*-gram language model's

fluency assessment of the translated sentence. Figure 1.3 gives an example of two candidate translations for the Chinese source sentence "這 [this] 間房子 [house] 我 [I] 喜 [like] 。". If all of the feature weights are set to 1.0, the highest scoring candidate translation would be "This house, I like." However, let's say the desired translation given in a reference sentence was "I like this house." From the system's perspective, the salient differences between the two candidates are that the first one, "This house, I like", has a lower language model score, while the second, "I like this house", has a higher linear distortion. MERT can use either of these differences to adjust the system's degree of preference for the two translations. As seen in figure 1.3, doubling the language model weight causes the second candidate to be the preferred translation. This change would improve the score assigned to the model if it was evaluated just over its ability to discriminate between these two sentences (i.e., effectively using one *n*-best list containing two items). However, to ensure that this change in model weights actually improves the performance of the system, we would need to rerun the decoder using the new model weights. This is to ensure that no new translation will be produced by the decoder using the new weights that both receives a higher decoding model score and a lower evaluation metric score than the preferred candidate, "I like this house."

1.2 Alternatives to MERT

While MERT is the most popular method for fitting a machine translation model to an evaluation metric, some work has been done using the margin infused relaxed algorithm (MIRA) (Chiang et al., 2009; Chiang et al., 2008; Watanabe et al., 2007). MIRA is an online large-margin learning algorithm (Crammer & Singer, 2003). Like other work with using discriminative learning algorithms for machine translation such as using averaged perceptron (Liang et al., 2006a), MIRA can approximate the fitting of an evaluation metric by using the metric to select the translations that will be used as the target of learning. However, unlike averaged perceptron, MIRA also allows the scores provided by the evaluation metric to be incorporated as the loss assigned to incorrect translations. While this work focuses on the use of MERT rather than MIRA, many but not all of the same issues apply. Within chapter 6, we will see that MIRA and other discriminative learning algorithms such as support vector machines for interdependent structured output spaces (Tsochantaridis et al., 2004), max-margin Markov networks (Taskar et al., 2004), and conditional log-likelihood training of log

linear models (Och & Ney, 2002) all perform similarly to MERT. Discussion will be provided when the results obtained using MERT may potentially differ from other learning methods.

1.3 Motivation

While not much work has been done on improving the parameterization of machine translation models, recently there has been increased interest in this area. There are, however, significant gaps in the literature such as: a comparison of different approaches to optimization; an investigation into the mapping between the choice of evaluation metric used for training and the actual quality of the resulting system; a method for training to more advanced but slower evaluation metrics; and a systematic comparison of MERT to other mainstream machine learning approaches.

Given the difficulty of directly optimizing an evaluation metric score during training, it is possible that there are dramatic performance differences between different algorithms that can be applied to the task. Recall that the objective is not smooth since small variations in the parameter values will usually not change the translations produced by the system and thus not affect the evaluation score. However, at certain critical points small variations in the parameters will change one of the translations produced and the assigned evaluation score. This problem has been approached using typical zero-order optimization techniques such as downhill simplex and Powell's method. However, the performance of the different techniques has not been systematically evaluated.

Directly fitting an evaluation metric also brings with it some risks. Most machine learning techniques include some smoothing or regularization component to ensure the resulting model will do reasonably well on not just the exact data that was used for training but also new unseen data. By exactly fitting an evaluation metric, it is possible that the resulting model will achieve a high evaluation score by exploiting something idiosyncratic about the training data and will not generally perform well. Prior work has not examined methods for regularizing the MERT objective.

Since the introduction of MERT, it has been generally assumed that improvements in the quality of automated evaluation metrics will directly result in improved machine translation systems (Och, 2003). Taking an automated measure that better reproduces human judgments of translation quality and then tuning a

system to maximize the score returned by the metric seems intuitively like it should result in a system that produces higher quality translations. Such reasoning also presumably holds for classifier based approaches that use evaluation metrics to select examples of good and bad translations to be used for training.

In practice, most machine translation systems are trained using the BLEU score (Papineni et al., 2002), a measure of *n*-gram overlap between the translation produced by the system and one or more reference translations. The popularity of BLEU for training is due in part to BLEU's use as the standard evaluation metric reported when publishing research results. BLEU's dominance was historically also due the lack of good alternative metrics.

Recently, however, the development of improved evaluation metrics has been a very active area of research. More recent metrics such as translation edit rate (TER) (Snover et al., 2006) and METEOR (Lavie & Denkowski, 2009) do correlate better with human judgments than BLEU. Since the DARPA GALE[1] project uses human translation edit rate (hTER) (Snover et al., 2006) to evaluate system quality, many systems developed for GALE have been tuned to TER as an approximation of hTER. Tuning systems to TER rather than BLEU results in systems that do achieve better TER scores. Since TER correlates better with human judgments, such systems also presumably produce translations that would be preferred by human judges. However, this assumption has not been explicitly tested.

Newer metrics such as METEOR and TER are being continuously enhanced. For example, an enhanced variant of TER, known as TER-plus (TERp), includes support for matching synonyms and paraphrases. While METEOR has always included support for matching synonyms, newer releases have also been extended to support paraphrase matches. These two metrics however are still fairly similar to BLEU in that they just focus on matching sequences of words in a reference translation to sequences of words in translations produced by a machine translation system. Matching more sequences and longer sequences will tend to result in the system being assigned a higher evaluation score by the metric.

No existing work has examined whether training to newer evaluation metrics like METEOR and TER actually results in better systems. By training to one of these newer metrics, it is expected that the resulting system will do better on the new metric than a similar system trained to BLEU. However, the most critical

[1] Defense Advanced Research Projects Agency: Global Autonomous Language Exploitation

question is *training to what metric results in the best translations according to humans?* Similarly, it also seems worthwhile to investigate how individual improvements in evaluation metrics affect model training. Knowing how or if certain improvements change the resulting system behavior would help inform what sorts of avenues for improvement are worthwhile with respect to training. For instance, recall that TER-plus allows for matching material in the reference translations to a machine translation using a paraphrase-table. While using a paraphrase-table improves TER-plus' correlation with human judgments, it is possible that it might only have a marginal impact on the quality of model training.

While there are other metrics that attempt to examine deeper structural relationships between references and machine translations, such metrics have not been used for training due to the amount of computation that is required. For instance, the headword chain based metric (HCBM) (Liu & Gildea, 2005), measures the syntactic similarity between two translations by matching dependency parse fragments. This allows for better scoring of translation pairs with surface level reordering that preserves the syntactic relationships between words. However, using this metric requires dependency parse trees, which can be time consuming to produce. Another metric, known as textual entailment evaluation (TEval) (Padó et al., 2009a), performs an even deeper level of analysis by using a collection of linguistically informed features that assess if two translations effectively mean the same thing. This metric is even more time consuming then HCBM since it requires both dependency parse trees, an alignment between the trees, as well as the output of numerous featurizers. Machine translation systems have not been tuned to such metrics since they are too slow to compute and MERT requires a large number of evaluation metric calls during training.

Using MERT brings with it some limitations in the types of decoding models that can be trained. For example, all of the typical optimization approaches used for MERT fail to scale well to larger numbers of decoding model parameters. When using line search methods such as coordinate descent or Powell's method to train a model with n parameters, the inner loop of training involves n expensive line searches and numerous executions of the inner training loop are needed to learn the best parameters for just a single set of n-best lists. Similarly, using downhill simplex to optimize a decoding model with n parameters requires n evaluation metric calls just to initialize the algorithm. Prior to convergence, simplex also requires numerous sets of n calls to the evaluation metric being optimized. This means that while MERT can be used to optimize

models with a dozen or so underlying parameters, it cannot be used to train models that contain millions or even thousands of decoding model features. This is unfortunate since using large numbers of binary features that fire for specific model states has been shown to be a very effective way of improving performance on other natural language processing tasks.

Since MERT restricts the space of decoding models we can use, it is important to verify that it actually is in fact a superior means of training machine translation models. When MERT was first introduced, Och (2003) demonstrated that MERT models vastly outperformed log-linear models trained to a conditional log-likelihood objective. However, recent work by Chiang et al. (2008) and Chiang et al. (2009) demonstrate that not only is the MIRA algorithm actually able to obtain the same level of performance as MERT when it is applied to a model containing a small number of features, but MIRA also is able to outperform MERT when it is used with a model containing over ten thousand features. This result suggests we should reevaluate whether MERT is in fact better than other mainstream machine learning methods. No systematic comparison exists that contrasts the performance of MERT to other popular machine learning algorithms.

In summary, prior work in this area has four significant problems. The first is that there has not been any thorough comparison of how well different optimization methods for MERT perform. Second, while it has been assumed that improved evaluation metrics will naturally produce better machine translations systems when they are used during training, this hypothesis has surprisingly never been explicitly tested. Third, many of the recently developed evaluation metrics all have a very similar character in that they still perform scoring based on how many overlapping word sequences appear in both a machine translation and a reference translation. Improvements in these evaluation metrics simply allow for fussier matching of semantically similar words and phrases. While more advanced automated metrics exist, training to them has not been previously explored. Finally, since recent work suggests that MERT is not a better method for training translation decoding models than the mainstream learning algorithm known as MIRA, it seems it would be worthwhile to reevaluate the presumed performance advantage of MERT over other popular algorithms from the machine learning community.

1.4 Contribution

This work explores the effectiveness of different methods for parameterizing machine translations systems. It contrasts the effectiveness of using different approaches to optimization for MERT. It compares existing methods to a new approach using randomized search directions and proposes a regularization heuristic for MERT. Both randomized search directions and regularization are found to improve the quality of the resulting models. This work also investigates the effects of tuning to different evaluation metrics and examines whether using improved evaluation metrics actually results in improved translation quality as determined by humans. As expected, tuning to different evaluation metrics results in systems that do better on the evaluation metric they were trained on than models trained to other metrics. However, it is found that when using metrics that essentially just measure word sequence overlap, such as BLEU, TER, and METEOR, human judges show little preference for models trained to any one metric over another. A technique for training a model to a more advanced but slower metric such as TEval is also presented. Finally, this work examines the effectiveness of training translation models using more mainstream machine learning methods including log-linear models, and a number of large-margin based methods.

1.5 Prior Work

Since the introduction of MERT, part of its theoretical appeal was that it allowed improvements in evaluation metrics to improve the quality of machine translation systems. However, no work has been done that investigates the link between evaluation metric quality and the quality of the resulting tuned machine translation systems. Part of the reason for this is that historically there were very few MERT tools that were publicly available. The ones that were available such as the CMERT packaged and its predecessor written in Perl, were hard coded to optimize the BLEU evaluation metric. Since important details of the MERT algorithm were only weakly described in Och (2003)'s original paper, research work on using new evaluation metrics with MERT required the non-trivial overhead of building the software tools necessary to perform optimization to the newer metrics. Similarly, this also reduced the amount of work on using different optimization algorithms for MERT.

1.5.1 Z-MERT

Zaidan (2009) introduced Z-MERT as a replacement for the popular CMERT package. Unlike CMERT, Z-MERT was designed to allow for pluggable evaluation metrics. Out of the box, Z-MERT supports optimization to BLEU, TER, and TER-BLEU. However, adding new evaluation metrics can be done by simply implementing an interface to the new metric using the Z-MERT API. Given the use of hTER as the official evaluation metric for the GALE project, Z-MERT's support of TER has made the package popular in the machine translation community.

One drawback of the package is that it does not provide out of the box support for the popular METEOR evaluation metric nor other newer metrics or metrics of historical significance such as PER (Tillmann et al., 1997) and m-WER (Nießen et al., 2000). The package also only provides one optimization algorithm, based on using coordinate descent in conjunction with the optimal global line search algorithm proposed in (Och, 2003).

1.5.2 Alternative Search Methods

The most common approach to MERT is to optimize model parameters through a series of line searches embedded in some larger search strategy such as coordinate descent or Powell's method. Zhao and Chen (2009) examine the use of downhill simplex for optimization. Their work found that downhill simplex tends to find better minima than line search approaches to MERT. The difference between the two approaches can be reduced by using more random restarts. However, even for a very large set of restarts, they found that downhill simplex still modestly outperformed line search approaches.

1.5.3 Tuning to METEOR

Since its introduction, there have been anecdotal claims that tuning to the METEOR evaluation metric results in poor system performance. The default parameterization of METEOR is recall orientated. It rewards the system translations more for including all of the words in the reference translation than it effectively punishes the system translation for including additional words not present in the reference translations. When

used for optimization, the claim was that METEOR resulted in system translations that were excessively long and that were more prone to hallucinating content that was not present in the original sentence that is being translated.

He and Way (2010) explored tuning a system to the default configuration of METEOR and confirmed much of the criticism of the metric when it is used to train a system. The translations produced by systems tuned to METEOR tended to be longer than equivalent systems tuned using BLEU. Additionally, while systems tuned to BLEU tend to error on the side of dropping content words, systems tuned to METEOR tended to insert additional words that were unsupported by the original sentence.

However, the METEOR metric includes three hyper-parameters that allow users to adjust the behavior of the metric. The α hyper-parameter controls how the metric trades off precision and recall during scoring and the β and γ hyper-parameters control how the metric trades off precision and recall with its penalty for word scrambling. One plausible way to improve METEOR's performance as a tuning metric would be to adjust how the metric trades off precision and recall. Increasing the weight the metric assigns to precision would presumable discourage systems tuned to it from producing excessively long translations that hallucinate new material not present in the original sentence.

He and Way (2010) found that adjusting the word scrambling penalty was actually a good method for improving the performance of systems tuned to METEOR. Increasing the extent to which system translations were punished for word scrambling not only encourages improved word ordering in the system translations, but it also implicitly discouraged the system from producing excessively long translations with hallucinated words. The reason for this is that under the default configuration, hallucinated words could sometimes be rewarded if METEOR could find somewhere in the reference translation words that could be associated with them. By turning up the word scrambling penalty, the system translations were heavily penalized for introducing any words that were not in the correct *position* in the reference translation. This heavily punishes translations that would otherwise receive high scores through random associations between words.

By increasing the γ chunking penalty, He and Way (2010) found that tuning systems to METEOR actually performed well even when being evaluated by the BLEU metric. In fact, systems tuned to this variant of the METEOR metric actually performed better on BLEU than a system specifically tuned to BLEU. This

later result is very interesting, since a priori one would expect a system to do best on whatever evaluation metric it was trained on and do slightly worse on other evaluation metrics when compared to systems that were specifically trained to the other metrics. This result seems to indicate that there are other factors at work such as how difficult the objective provided by different evaluation metrics is to optimize. For example, the objective provided by one evaluation metric maybe prone to containing more local minima that another.

1.6 Organization

The next chapter describes the Phrasal machine translation software that was developed to perform the experiments presented in this work. Phrasal is a state-of-the-art phrase-based machine translation package. It is similar to Moses and even supports the use of many Moses decoding models. The package provides an easy to use feature API for the creation of new decoding model features and a flexible implementation of MERT that allows for training to all popular machine translations evaluation metrics using a variety of approaches to optimization.

In the subsequent chapter, Phrasal is used to explore the effectiveness of different approaches to optimization for MERT. Results are presented for Powell's method and coordinate descent. The chapter also presents a new method based on performing line searches along randomly chosen search directions. Random search directions are found to outperform Powell's method and coordinate descent. This chapter also explores a heuristic for smoothing the objective function optimized during MERT by averaging the evaluation metric score assigned to adjacent plateaus within the objective. The heuristic is shown to improve the performance of Powell's method and coordinate descent. However, combining the smoothing heuristic with random search directions leads to no further improvements over what is achieved when either approach is used in isolation.

Chapter four examines the properties of decoding models trained to different popular evaluation metrics. It investigates the performance of systems trained to one metric and then evaluated on another and the stability of models trained to the different metrics. The chapter closes by examining the connection between the evaluation metric used during training and actual human preferences for the translations produced by the resulting systems. It is found that the BLEU score has a number of desirable properties when it comes to

training. BLEU trained models perform reasonably well when evaluated by other evaluation metrics. The models are also more stable, with there being less variation in performance across different training runs. Finally, while there was no clear preference for one evaluation metric over another when the models trained to them were evaluated by human judges, the BLEU trained models tended to do reasonably well for both language pairs examined. With other metrics like TER and METEOR, some configurations of the metrics did okay on one language pair, but less well on another.

Chapter five investigates the possibility of training to the Stanford entailment metric, a sophisticated evaluation metric that performs a deep analysis of the translation being evaluated and its reference. Unlike word sequence overlap metrics like BLEU, TER, and METEOR, this metric uses a collection of syntactic and semantic features to determine whether a machine translation and a reference translation mean the same thing. In its original form, this metric is very slow taking seconds to evaluate each pair of sentences presented to it. The chapter explores methods for speeding up the metric and using a staged training regime involving first training a system to a faster metric, such as BLEU or TER, and then training to TEval using smaller *n*-best lists.

Chapter six contrasts the performance of MERT with several mainstream machine learning methods including conditional likelihood training of log-linear models, max-margin Markov networks, support vector machines for interdependent and structured output spaces, and MIRA. The results show these learning methods are just as effective as MERT. This equivalence may seem to contrast with the results found in Och (2003)'s work that introduced MERT, since in his work MERT trained models were generally shown to *overwhelmingly* outperform conditional likelihood training of log-linear models. To account for this discrepancy, I provide an analysis that reconciles Och (2003)'s results with my own. From the analysis and the experimental data, I conclude that MERT does not actually have a performance advantage over other state-of-the-art machine learning methods. Finally, chapter seven concludes and discusses possible directions for future work.

Chapter 2

Phrasal: A Toolkit for Statistical Machine Translation

2.1 Introduction

Later chapters will describe experiments on using different optimization techniques and evaluation metrics for MERT. When this work first began, there were no existing software packages that could be used to perform such experiments. David Chiang's CMERT[1] package is hardwired to perform optimization using coordinate descent and to use BLEU as the evaluation metric being optimized. Inspection of the software's source code revealed that it did include an orphaned implementation of Powell's method, but it is not available to users of the package without making source code changes. The later introduction of the Z-MERT package (Zaidan, 2009) allows for swappable evaluation metrics. However, out of the box Z-MERT supports only a subset of the well known evaluation metrics used by the machine translation community.

While this work does not directly investigate the use of new decoding model features, many of the results that are presented strongly suggest future work should include feature engineering research. For example, more sophisticated decoding model features could allow us to better take advantage of evaluations metrics like the Stanford entailment metric that perform deeper linguistic analysis of the machine and reference translations. There are a number of phrase-based machine translation packages. However, the existing packages are not designed around being used for decoding model feature engineering. The Pharaoh machine translation system (Koehn, 2004) is closed source, so it is not possible to add new decoding model features, unless they can be included as additional scores for each phrase pair in the phrase-table. This restricts

[1] The CMERT package was released informally by David Chiang. But, it became broadly used after it was incorporated into the Moses machine translation package

new features to just those that effectively provide a new way of scoring phrase pairs and there is no way to add a feature that scores across phrases. The Phramer package (Olteanu et al., 2006) is an open source re-implementation of Pharaoh in Java. While it would be possible to use this package for feature engineering, doing so would require substantial changes to the code since it very strongly assumes that the 8 Pharaoh decoding model features are the only features that will be used. The Moses package (Koehn et al., 2007) is an enhanced re-implementation of Pharaoh that allows for the use of factored decoding models. This allows for the introduction of new decoding model features that can either be represented as a translation or language model score for an abstract "factored" representation of the phrases being translated (e.g., phrases consisting of POS tags) as well as features that score the combination of tokens across factors. While, this does enhance the space of features that can be developed and used with the Moses decoder, it still does not allow for flexible feature engineering outside the general kind of features that are typically used for phrase-based machine translation systems. Similar to Phramer, earlier versions of the Moses code strongly assumed that the features that were going to be used on the factors were fixed and included hard coded calls to the individual features. Later versions of the code do however include a more flexible implementation of the code involved with features.

In order to perform the desired experiments and to facilitate future related research, I developed Phrasal (Cer et al., 2010b), a state-of-the-art phrase-based machine translation system. As originally developed, the system provides what I believe were the most useful attributes of existing systems in addition to an API that allows for easy feature engineering. This facility for feature engineering has been used to develop new innovative features. These features allow Phrasal machine translation models to outperform other similar phrase-based system and even hierarchical machine translation systems. The package's implementation of MERT allows for pluggable evaluation metrics and supports optimization using all popular evaluation metrics used by the machine translation community. It also supports optimization using a variety of algorithms including coordinate descent, Powell's method, and downhill simplex.

The toolkit was implemented in Java, since Java offers a good balance between performance and developer productivity. Compared to C++, developers using Java are 30 to 200% faster, produce fewer defects, and correct defects up to 6 times faster (Phipps, 1999). While Java programs were historically

much slower than similar programs written in C or C++, modern Java virtual machines (JVMs) result in Java programs being nearly as fast as C++ programs (Bruckschlegel, 2005). Java also allows for trivial code portability across different platforms.

The next section presents the beam search decoding algorithm used by Phrasal and other phrase-based machine translation systems. After that, I review various important implementation details necessary for obtaining good translation performance as well as unique software engineering issues that arise when implementing a machine translation system in Java. The chapter concludes by presenting the usage of Phrasal to build new machine translations system, unique attributes of the system, and a highlights of other research that was enabled by the creation of the software package.

2.2 Beam Search Decoding for Phrase-Based Machine Translation

Phrasal uses a left-to-right beam search decoding algorithm that was first introduced with the Pharaoh system (Koehn, 2004). The same algorithm is also used by both the Moses (Koehn et al., 2007) and Phramer (Olteanu et al., 2006) systems. The algorithm operates by selecting and translating words and phrases from the original source sentence and then appending their translation to the end of the translation being constructed. The search is guided by a decoding model that evaluates the quality of the translations being constructed according to their apparent preservation of meaning of the words and phrases that are translated, the appropriateness of reorderings of translated material from their relative position in original sentence to their relative position in the translation, and the fluency and grammaticality of the translation. A beam search is used since choices that appear good locally may ultimately lead to a lower quality translation. Within the beam search, hypotheses compete with each other based upon the sum of the scores obtained from making local decoding decisions and an estimate of the future cost of covering material in the original sentence that is not yet address by the partial translation. The future cost estimate critically allows for better competition between hypotheses that have already translated a very difficult part of the original sentence with hypothesis that have only addressed the portions of the original sentence that are relatively easy to translate.

2.2.1 Hypothesis Construction

Figure 2.1,[2] illustrates the process of constructing a single translation within a left-to-right phrase based machine translation system. Initially, the system starts out with an empty hypothesis that covers none of the material in the original sentence. The system then selects a word or continuous span of words in the original sentence to translate. It then translates the word or phrase,[3] and constructs a new more complete working hypothesis by appending the material just translated to the end of the partial translation for the rest of the sentence. Since the complete translation is constructed by incremental concatenation of newly translated material, the process is traditionally known as left-to-right phrase-based translation. However, a more technically correct term maybe concatenative phrase-based machine translation, since when translating into languages like Arabic, Hebrew, or Farsi the translations can actually be seen as being built up from right-to-left.

When a new partial hypothesis is constructed it can be characterized by four defining factors. The first is the string that represents the most recently translated word or phrase. The second is a score corresponding to the merit of the hypothesis, which is used by the decoding algorithm to determine which hypotheses are worth extending and which ones are of sufficiently low quality that they are unlikely to lead to good translations of the original sentence. The merit score is computed by summing the score the decoding model assigned the parent of the new hypothesis plus the score from the decoding model for the most recent extension of the hypothesis, and an estimate of the cost associated with translating material in the original sentence that is not yet accounted for the by partial translation. The third defining factor is a bitmap that marks what words in the original sentence have been accounted for by the partial translation and what positions remain to be translated. When the decoder constructs new hypotheses from existing ones, this bitmap is consulted to determine what positions in the original sentence are still available for translation. When all of the positions in the source sentence are covered, the decoder has reached a complete translation of the original sentence. Finally, the last defining factor of a hypothesis is a back pointer to the parent hypothesis from which it was

[2] Figure reproduced from Chapter 1.

[3] Recall that in phrase-base machine translation, the term “phrase” is used very loosely to refer to any adjacent set of words, rather than in any proper linguistic sense of the word phrase. As we will see later, recent extension to Phrasal have allowed this constraint to be further loosen, with phrases now being able to contain gaps in the middle of them.

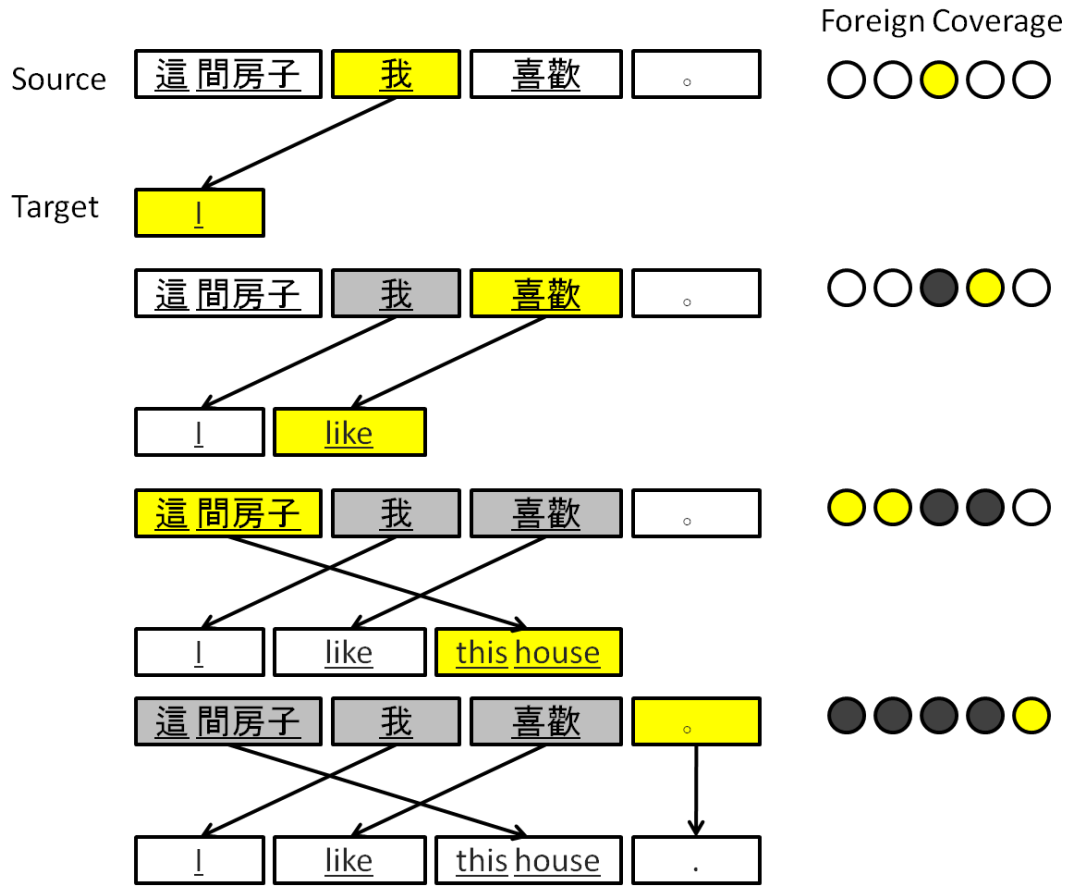

Figure 2.1: **Phrase-Based Decoding (Reproduced from Chapter 1):** The translation is produced from left-to-right by selecting and then translating words and phrases in the original sentence. In addition to the current partial translation, the decoder keeps track of what material has been translated so far as well as the location of the last foreign material that was translated.

constructed. This allows the decoder to compute certain decoding model scores that depend on knowing the structure of the existing partial translation. Typically, the model scores that use this information are those that look at the reordering of material during translation. For example, the linear distortion feature and hierarchical reordering features needs to know the position in the original sentence of the phrase that was translated by the parent hypothesis.

In the interest of performance optimization, Phrasal and other phrase-based decoders cache some information that could be derived by simply inspecting the back pointers to parents and ancestors of a hypothesis. For example, decoders can cache the position in the original sentence of the end of the phrase that was just translated. This allows linear distortion to be calculated very efficiently using just the expression

$\mathrm{distortion} = \text{-abs}(\mathrm{endPosPriorTranslatedPhrase}\text{-posPhraseBeingTranslated}+1)$. If each hypothesis did not store the end position in the original sentence of the most recently translated phrase, $\mathrm{endPosPriorTranslatedPhrase}$ would need to be computed by XOR-ing the bitmaps of a hypothesis' parent and the parent's parent and then searching for the position of the most significant non-zero bit. Similarly, under some configurations, Moses style lexical reordering features need to know the identity of both the source and the target sides of the phrase that was translated by the parent hypothesis as well as the source and the target side of the phrase that was just translated by the current hypothesis. While such information could be reconstructed by inspecting bit-sets and following back pointers, it is far more computationally efficient to just cache this information explicitly in the hypothesis. As will be discussed later in this chapter, since one of the design goals of the Phrasal machine translation system was to export an API that would allow for easy engineering of new knowledge sources for the decoding model, Phrasal caches a significant amount of information in each hypothesis that is constructed including a representation of the complete partial translation associated with the hypothesis, and information regarding the alignment of each translated phrase to its position in the original sentence.

2.2.2 Beam Search

Typical implementations of beam search in other domains proceed as follows. The system first constructs an empty hypothesis corresponding to the system's start state. The system then uses the search operates that are available to it to expand the hypothesis. This produces new hypotheses and often numerous new hypotheses. These new hypotheses are then scored by some figure of merit that is appropriate for the domain. For example, a path finding problem might use the distance covered by each hypothesis and some approximate estimate of how far away the partial solution is from reaching the goal position. The system then sorts the hypotheses by this figure of merit and only keeps either the top stack_{max} highest scoring hypotheses or the hypotheses that have a figure of merit that is at most δ_{max} worse than the hypothesis with the best figure of merit. Pruning to only the top n best hypotheses is known as histogram pruning, while pruning based on the maximum distance between the figures of merit is known as threshold pruning.

Beam search implementations may incorporate some form of *recombination* whereby the system keeps

track of which hypotheses are necessarily worse than other hypotheses that have already been constructed given the structure of the domain. Returning to the example of a path finding problem, recombination could take the form of keeping track of the lowest cost found so far for each position on the map. If a new hypothesis is constructed and it arrives at a position that has been previously discovered by a lower cost hypotheses, then the newer more costly hypothesis can be safely discarded since there is no way that its descends will be able to reach the goal with a lower cost than those of the previously constructed lowest cost hypothesis for the current position. When using threshold pruning, recombination can open up a significant number of positions on the beam and allow the space to be searched much more thoroughly than would have been possible using the same beam size and no recombination. When the beam is being pruned by threshold pruning, using recombination can help to speed up the search by not wasting time exploring hypotheses that, while still being within δ_{max} of the best hypothesis, will necessarily give rise to lower scoring descends than alternatives.

Phrasal aggressively recombines hypotheses by looking inside of the standard decoding features and making use of subtle differences in what is and is not captured by the feature. For example, a naive implementation of a recombination rule for a system that includes an *n*-gram language model might not allow hypothesis recombination if the last *n*-1 words in the partial translations associated with two hypotheses are different, where *n* is the order of the language model being used. However, the recombination rules for both Phrasal and other decoders such as Moses actually inspect the contents of the language model being used and allow the recombination of hypotheses with matching suffixes that are much shorter than order of language model. This is done by looking up the longest suffix of each partial translation that can be found in the *n*-gram language model. For the purposes of the language model score, translations with identical longest match suffixes can then be safely recombined. This approach exploits the fact that higher order *n*-grams are typically very sparse. Even though say a 5-gram language model is being used, the language model often computes a score by backing-off to lower order *n*-grams.

The quality of the solution found by a beam search is determined by three factors.[4] The first is the

[4] For tasks like machine translation, there are two senses of the quality of the solution. The first is the quality of the solution according to the decoding model that is used to drive the search. This is the sense of "quality" that is used here. Another is the actually quality of the resulting translation which depends on how well the scores returned by the decoding model actually reflect

size of the beam as defined by stack_{max}, δ_{max}, or both. The larger the maximum stack size and the further the search is able to deviate from the figure of merit assigned to the current best hypothesis, the more thoroughly it will search the space and the more likely it will be to find a better lower cost solution. Second, better future cost estimates allow the algorithm to more precisely search the hypothesis space and avoid exploring regions that are unlikely to result in good quality solutions. In the extreme case, a perfect future estimate would allow the search to use a beam that only ever contains one hypothesis, e.g., $\text{stack}_{max} = 1$ or $\delta_{\text{max}} = 0$, and still proceed directly to the goal state along the lowest cost route. Alternatively, a very poor future cost estimate may systematically encourage the search to waste time, and more importantly space on the beam, exploring regions of the hypothesis space that are unlikely to lead to good quality solutions. Third, as previously mentioned, the existence of recombination allows the search to explore much more of the hypothesis space. For complex search problems like machine translation, developing correct but aggressive recombination rules requires knowing exactly what knowledge sources and features are being used by the decoding model. An admissible recombination rule can then be derived by only allowing hypotheses to be recombined if the differences between them could not possibly affect the scores that will be returned by any of the features in the future.

2.2.3 Future Cost Estimate

Future cost estimation for phase-based machine translation is performed by running the decoding model features over the individual phrases that could be used to translate a source sentence. The locally featurized phrases are then scored using the decoding model's feature weights. Using these scored phrases, the system searches for the lowest cost selection of phrases that cover each part of the source sentence. The result is a future cost matrix that specifies the minimum cost of covering any continuous span of words in the source sentence. When calculating the future cost of a hypothesis, the decoder sums over the future cost matrix entries for all of the uncovered spans of words in the source sentence that still need to be translated.

---

the true quality of the translations being constructed.

2.2.3.1 Features used for future cost

The decoding model features used for the future cost matrix include all of the features that are not used to score reorderings during translation. The complete set of features used by default for the future cost matrix is given below:

- *n*-gram language model(s)
- Phrase-table translation probabilities
- Phrase penalty, word penalty and unknown word penalty

Of these, all but the *n*-gram language model score(s) can be calculated exactly at the level of individual bilingual phrase pairs. The *n*-gram language model score used for the future cost heuristic is obtained by running the language model over just the target set of each phrase pair. Usually, the decoder is run with a language model that was trained on the same data that was used to extract the bilingual phrase pairs. Running the language model on the target side of each phrase pair is then running the language model on the same data that it was trained. This and the fact that the language model is not applied across phrase boundaries when calculating the future cost causes the language model component of the future cost heuristic to tend to under estimate the true cost of completing the hypothesis.

However, the estimate is not strictly lower than the true cost. It is possible to construct scenarios containing words that are very unlikely to occur in general, and thus have low language model scores, but that have a very high probability of occurring in certain contexts. For example, “ado” is a relatively rare word in English and in isolation it will receive a low language model score. However, in the context of “Without further ado” it has a much higher probability of occurring.[5]

Not accounting for the re-ordering features also causes the future cost matrix to tend to underestimate costs. Such underestimation makes it so that it would not be reasonable to compare hypotheses at different levels of completion when pruning the beam. Mostly complete hypotheses would have accurate figures of

[5] In practice, however, if the system knows the word “ado”, then the phrase “Without further ado” would also be available in the phrase-table. When calculating the future cost heuristic, having the whole idiomatic expression captured within a single phrase would allow the language model to score “ado” in context and thus not overestimate the cost of translating the larger span.

merit that are mostly based on the actual score assigned by the decoding model to the hypothesis. Largely incomplete hypotheses would have grossly inflated figures of merit based mostly on underestimated future cost estimates. Such differences in hypothesis completion are a real risk during phrase-based machine translation, since some hypothesis expansions can translate as little as one more word of the source sentence, while others can translate a larger multi-word phrase. Section 2.2.4 discusses how this issue is addressed within the decoder by using different beams for hypotheses at different levels of completion.

2.2.3.2 Computing future cost

The algorithm to compute the future cost for any continuous span is somewhat similar to bottom-up chart parsing. First, the system searches for the lowest cost way of translating each individual word in the source sentence. Then, the system proceeds to search for the lowest cost covering of all spans of two consecutive words. This is done by first consulting with the phrase-table to find the lowest cost translation for each two word span. The cost of covering the span with a two word entry from the phrase-table is then compared to the cost of covering the span using the lowest cost covering of the two individual words that it contains. That is, the system checks whether the cheapest 2-word entry from the phrase-table is less expensive than the sum of the cheapest 1-word entries that can be used to cover the same span. As shown in figure 2.3, the algorithm continues like this until it constructs a span that covers the whole sentence.

For higher order spans, there are numerous possible covering segmentations as seen in that, for a span of length n, there are $O(2^{n-1})$ different ways to cover it with smaller spans. It is, however, easy to check all of these efficiently. To do so, the algorithm only needs to check all possible divisions of a span into two chunks and, if it is available, compare the lowest cost of these to the cheapest entry from the phrase-table that could be used to cover the span. The future cost matrix will have already computed the lowest cost covering of each span that is smaller than the one that it is currently computing. By comparing all two-span coverings, the algorithm is performing the comparison using the lowest cost sub-segmentation of each of the two spans. For example, in figure 2.3, computing the future cost for the span [間房子 我 喜歡] by examining the future cost for the span [間房子] and [我 喜歡] implicitly is using the lowest cost sub-segmentation of the two word span [我 喜歡].

Figure 2.2: **Future Cost Estimation** : Future cost estimation computes the lowest cost covering of each span of consecutive words in the input sentence. The cost of covering a span is either the cheapest covering using a single entry from the phrase-table, or the cheapest covering achieved by segmenting the span into smaller spans that are in turn covered by entries from the phrase-table. The computation of the future cost matrix is done using dynamic programming and is somewhat similar to bottom up chart parsing. First, the algorithm computes the lowest cost covering of all spans of length 1. Then, it computes the lowest cost covering of all spans of length 2 by comparing the lowest cost coverings found from the 1-spans to the lowest cost covering of the 2-spans using entries from the phrase-table. The algorithm continues until it reaches a span that covers the entire sentence.

Pseudo-code for the future cost calculation can be seen near the top of algorithm 1. Computing the future cost estimates is $O(n^3)$ in the length of the source sentence being translated. Using this future cost estimate makes the time complexity of left-to-right beam search decoders $O(n^3)$, which actually matches that of hierarchical decoders which explicitly use CKY for decoding (Chiang et al., 2005). However, while the time complexities are the same, the future cost estimate for left-to-right beam search decoders executes *much faster* than hierarchical decoders.

2.2.4 Partitioning Beams by Source Coverage

Applying the traditional formulation of beam search to phrase-based machine translation would result in the system comparing hypotheses at drastically different levels of completion. For example, starting with an empty hypothesis, let's say that the search algorithm ran for two iterations. During each iteration, it loops through all of the existing hypotheses and expands them with whatever word or phrase translations

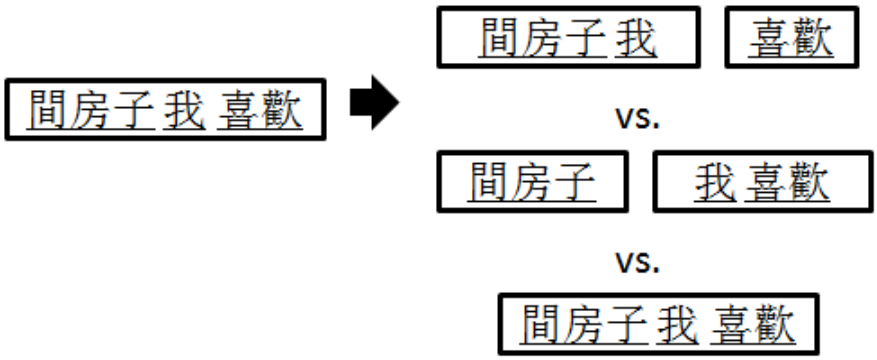

Figure 2.3: **Future Cost Comparison**: Computing the lowest cost covering of higher order spans can be done efficiently by reusing the lowest cost covering already computed for lower order spans. In this example, the lowest cost covering for [間房子我喜歡] is computed by comparing the lowest cost covering of [間房子] and [我喜歡]. The lowest cost covering of [我喜歡] has already performed the analysis of whether it is cheaper to cover [我喜歡] as a single chunk using an entry from the phrase-table or whether it would be less expensive to cover the span using translations for the individual words [我] [喜歡]

that can be applied. Doing two iterations would mean that the least complete partial hypotheses would only cover two words in the source sentence being translated. The most complete hypotheses could cover up to $2*\text{maxPhraseLength}$ positions, where maxPhraseLength is the maximum length of the source side of phrases in the phrase-table. In real systems, this is typically a value of about five. Consequently, after only two iterations, the system may have some hypotheses on the beam that cover only a small fraction of the larger sentence being translated, while others are almost complete translations.

Having such drastically different hypotheses completion for space on the same beam increases the risk that a good hypothesis will get squeezed out, since the future cost estimate tends to underestimate the cost of completing each hypothesis. The approach taken by beam search decoders to address this problem is to have separate beams for hypotheses at different levels of completion. For example, one approach would be to have a separate beam for each unique coverage bit map. This would mean that hypotheses would only be competing with other hypotheses that were making use of the exact same future cost estimate. However, such an approach scales poorly since the number of unique coverage sets is 2^n for a sentence of length n.[6]

The approach that is taken by Phrasal and other decoders such as Moses, Phramer, and Pharaoh is to

[6] One way to make decoding with a separate beam for each coverage set tractable would be to use a very tight distortion limit. Using a distortion limit of l and decoding a sentence of length n would require only $O(n2^l)$ beams.

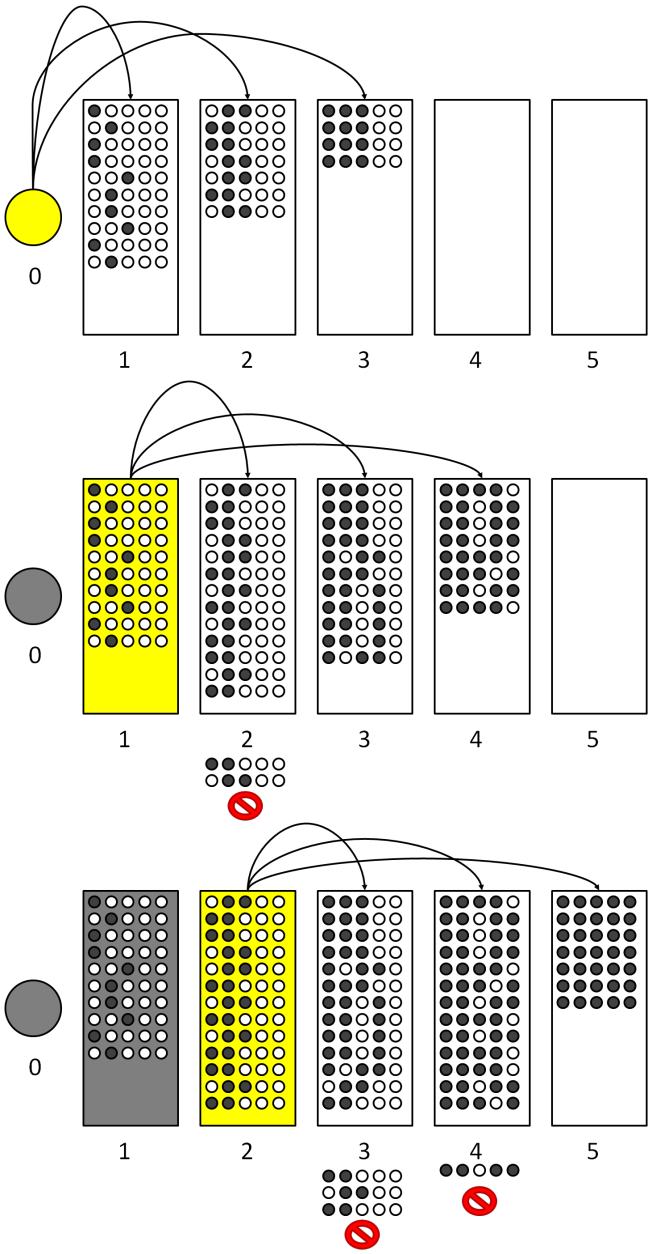

Figure 2.4: **Beam-Search Decoding**: The decoder starts with an empty hypothesis that covers none of the words in the source sentence. This hypothesis is expanded by selecting and translating a single span of source words. Each new hypothesis that is expanded is placed on a subsequent beam based on how many source words are now covered in the hypothesis. After the decoder exhausts all possible expansions of the empty hypothesis, it starts to expand the hypotheses on the $\mathrm{beam}[1]$, which corresponds to partial translations that cover one source position. After expanding all of those hypotheses, the decoder continues to process the beams in order until it reaches the last beam on which all hypotheses cover all source positions and are thus complete translations of the source sentence. The decoder then selects the highest scoring hypothesis from this beam and returns it as its best guess for the translation of the original sentence.

have a separate beam for each number of positions covered in the source sentence being translated. As illustration in figure 2.4, the decoder operates as follows. The system begins by expanding the empty hypothesis by translating phrases of various lengths in the source sentence. If the translated phrase only consists of a single word, the corresponding newly constructed hypothesis is then added to the beam for hypotheses that only cover one word in the source sentence. While the hypotheses on this beam all only cover one word, the actual location of the covered word can vary from one hypothesis to another. Since one hypothesis may have translated a particularly difficult to translate word, while another covered one with a relatively straightforward translation, the hypotheses depend on the future cost estimate to ensure that the competition between them is fair. Other expansions of the empty hypothesis may cover more source positions, up to the longest source side entry in the phrase-table that can be applied. The new hypotheses that cover *n* source positions are all added to the *n*-th beam.

After all possible expansions of the initial empty hypothesis are complete, the system begins to expand hypotheses on $\mathrm{beam}[1]$, the beam corresponding to hypotheses that all cover at least one source position. After all of the hypotheses on $\mathrm{beam}[1]$ have been completely expanded, the system moves on to expanding hypotheses on $\mathrm{beam}[2]$. This process continues until the system reaches the last beam in the series, with hypotheses on this beam all corresponding to complete translations of the source sentence. The system can then extract the highest scoring hypothesis on this beam and return it as the most appropriate translation of the source sentence.

2.2.5 Pseudo-code

Pseudo-code for the complete decoding algorithm is given in algorithm 1. The decoder starts by retrieving all bilingual phrase pairs from the phrase-table that could be used to translate a word or phrase in the source sentence. The decoding model then scores each phrase pair using all of the model features that can be applied to a phrase pair in isolation (i.e., *n*-gram language model score of the target side of the phrase pair, the phrase pair translation probabilities, the phrase penalty, the word penalty, and unknown word penalty). Given the portions of the source sentence each phrase pair can be used to translate, the resulting isolated phrase pair scores are then used to calculate the highest scoring way of covering each possible span of text

Algorithm 1 Left-to-Right Phrased-Based Decoding: The left-to-right phrase-based decoding algorithm first introduced by the Pharaoh machine translation system and that is also used by both Moses and Phrasal.

```
Input: f: source sentence, wts: weights, pt: phrase-table, fts: features, isofts: isolated phrase features
// Compute Phrase Coverage Chart M
for i = 0 to f.length-1 do
   for j = i to min(f.length-1,i+pt.maxPhraseLen) do
      M[i,j] = pt.getTranslations(f[i,j])
   end for
end for
// Compute Future Cost Estimate Matrix F
for span = 1 to f.length do
   for i = 0 to f.length-span do
      F[i,span-1] = min(phr ∈ M[i,span-1] : wts.score(isofts.featurize(phr)))
      for j = i+1 to i+span-1: do
         F[i,span-1] = min(F[i,span-1], F[i,j]+F[j,span-1])
      end for
   end for
end for
beam[0] ← empty hypothesis
// Main decoding loop
for b = 0 to f.length: do
   for h ∈ beam[b] do
      for i = 0 to h.lastUncover + distortionLimit do
         for j = i to f.length do
            for phr in M[i,j] do
               newH ← h.appendPhrase(phr)
               if RecombinationHashBest[newH] > newH.merit break
               RecombinationHashBest[newH] = newH.merit
               beam[newH.coveredCount].addThenPrune(newH)
            end for
         end for
      end for
   end for
end for
// Return highest scoring hypothesis
e ← beam[f.length].best()
return e
```

in source. The score associated with the best way to cover each span then becomes the future cost estimate for that span. With the future cost estimates in hand, we are now ready to enter the main decoding loop. The decoder first constructs an empty hypothesis that covers none of the words in the source sentence and places this hypothesis on the first beam. As described above, it then proceeds to loop through the beams one-by-one expanding hypotheses and then placing their children on the appropriate downstream beam for subsequent processing. Once all beams have been processed, the decoder selects the best scoring hypothesis from the last beam, which recall contains the hypotheses that cover all source sentence positions. This hypothesis is returned as its best guess for the appropriate translation of the original sentence.

2.3 Toolkit Usage

The toolkit provides easy to use end-to-end support for the creation and evaluation of machine translation models. Given sentence-aligned parallel text, a new translation system can be built using a single command:

```
java edu.stanford.nlp.mt.CreateModel \
    (source.txt) (target.txt) \
    (dev.source.txt) (dev.ref) (model_name)
```

Running this command will first create word level alignments for the sentences in source.txt and target.txt using the Berkeley cross-EM aligner (Liang et al., 2006b).[7] From the word-to-word alignments, the system extracts a phrase-table (Koehn et al., 2003) and hierarchical reordering model (Galley & Manning, 2008). Two *n*-gram language models are trained on the target.txt sentences: one over lowercased target sentences that will be used by the Phrasal decoder and one over the original source sentences that will be used for truecasing the machine translation output. Finally, the system trains the feature weights for the decoding model using minimum error rate training (Och, 2003) to maximize the system's BLEU score (Papineni et al., 2002) on the development data given by dev.source.txt and dev.ref. The toolkit is distributed under the GNU general public license (GPL) and can be downloaded from: http://nlp.stanford.edu/software/phrasal

[7] Optionally, GIZA++ (Och & Ney, 2003) can also be used to create the word-to-word alignments.

2.4 Advanced Features

Phrasal includes a number of advanced features the make it an attractive choice for machine translation research. These include multiple decoding engines, good multi-threading support, and an easy to use feature engineering API.

2.4.1 Decoding Engines

The package includes two decoding engines, one that implements the left-to-right beam search algorithm that was first introduced with the Pharaoh machine translation system (Koehn, 2004), and another that provides a recently developed decoding algorithm for translating with discontinuous phrases (Galley & Manning, 2010).[8] Both engines use features written to a common but extensible feature API, which allows features to be written once and then loaded into either engine.

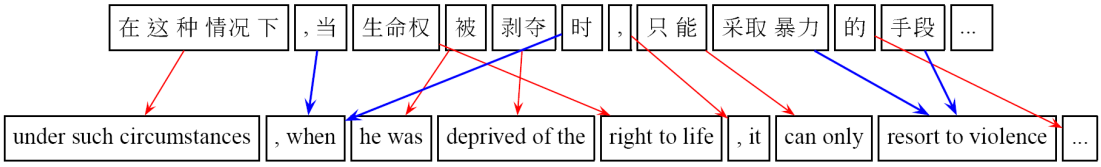

Figure 2.5: **Chinese-to-English Translation using Discontinuous Phrases:** Translating with discontinuous phrases allows a *single* phrase to cover non-adjacent words in the source sentence as well as for a single phrase to produce non-adjacent words in the resulting translation. This allows the system to elegantly translate grammatical constructions that encircle other material such as the Chinese 当 **X** 的 that can be translated as *when* **X** or the French negative construction *ne pas*.

Discontinuous phrases provide a mechanism for systematically translating grammatical constructions. As seen in Fig. 2.5, using discontinuous phrases allows us to successfully capture that the Chinese construction 当 **X** 的 can be translated as *when* **X**.

2.4.2 Multi-threading

The decoder has robust support for multi-threading, allowing it to take full advantage of modern hardware that provides multiple CPU cores. As shown in Fig. 2.6, decoding speed scales well when the

[8] The discontinuous phrase decoding engine was written by Michel Galley by modifying the original left-to-right beam search decoder.

number of threads being used is increased from one to four. However, increasing the threads past four results in only marginal additional gains as the cost of managing the resources shared between the threads is starting to overwhelm the value provided by each additional thread. Moses also does not run faster with more than 4-5 threads.[9]

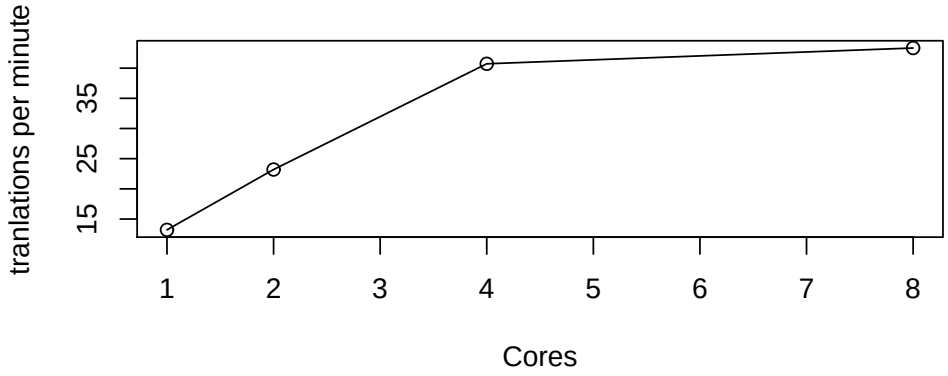

Figure 2.6: **Multi-Core Performance:** Multi-core translations per minute on a system with two Intel Xeon L5530 processors running at 2.40GHz.

2.4.3 Feature API

The feature API was designed to abstract away complex implementation details of the underlying decoding engine and provides a simple consistent framework for creating new decoding model features. During decoding, as each phrase that is translated, the system constructs a *Featurizable* object. As seen in Table 2.1, *Featurizable* objects specify what phrase was just translated and an overall summary of the translation being built. Code that implements a feature inspects the *Featurizable* and returns one or more named feature values. Prior to translating a new sentence, the sentence is passed to the active features for a decoding model, so that they can perform any necessary preliminary analysis.

2.5 Comparison with Moses

Credible research into new features requires baseline system performance that is on par with existing state-of-the-art systems. Seen in Table 2.2, Phrasal meets the performance of Moses when using the exact

[9] http://statmt.org/moses/?n=Moses.AdvancedFeatures (April 6, 2010)

Featurizable	
Last Translated Phrase Pair	Source and Target Alignments
Partial Translation	Source Sentence
Current Source Coverage	Pointer to Prior Featurizable

Table 2.1: **Featurizable:** Information passed to the decoding model features in the form of a *Featurizable* object for each new translation hypothesis built by the decoder.

same decoding model feature set as Moses and outperforms Moses significantly when using its own default feature set.[10]

System	Features	MT06 (tune)	MT03	MT05
Moses	Moses	34.23	33.72	32.51
Phrasal	Moses	34.25	33.72	32.49
Phrasal	Default	35.02	34.98	33.21

Table 2.2: **Phrasal vs. Moses:** Comparison of two configurations of Phrasal to Moses on Chinese-to-English. One Phrasal configuration uses the standard Moses feature set for single factor phrase-based translation with distance and phrase level msd-bidirectional-fe reordering features. The other uses the default configuration of Phrasal, which replaces the phrase level msd-bidirectional-fe feature with a hierarchical reordering feature.

2.6 Other Research Enabled by Phrasal

The unique features of Phrasal have supported a number of research projects other than my own work reported in this dissertation. The following is a summary of selected work that has made use of Phrasal. It includes the development of three new re-ordering models, and improved measures of translation fluency using a dependency language model.

2.6.1 Hierarchical Phrase Reordering Model

Galley and Manning (2008) uses the toolkit to develop a hierarchical generalization of the Moses lexical reordering model. Instead of just looking at the reordering relationship between individual phrases,

[10] Phrasal was originally written to replicate Moses as it was implemented in 2007 (release 2007-05-29), and the current version still almost exactly replicates this implementation when using only the baseline Moses features. To ensure this configuration of the decoder is still competitive, we compared it against the current Moses implementation (release 2009-04-13) and found that the performance of the two systems is still close. The current Moses implementation obtains slightly lower BLEU scores, respectively 33.98 and 32.39 on MT06 and MT05.

the new feature examines the reordering of blocks of adjacent phrases and improves translation quality when the material being reordered cannot be captured by a single phrase. This hierarchical lexicalized reordering model is used by default in Phrasal and is responsible for the gains shown in Table 2.2 using the default features.

2.6.2 Target Side Dependency Language Model

The *n*-gram language models that are traditionally used to capture the syntax of the target language do a poor job of modeling long distance syntactic relationships. For example, if there are a number of intervening words between a verb and its subject, *n*-gram language models will often not be of much help in selecting the verb form that agrees with the subject. Galley and Manning (2009) develop a target side dependency language model feature that captures these long distance relationships by providing a dependency parse score for the target translations produced by the decoder. This is done using an efficient quadratic time algorithm that operates within the main decoding loop rather than in a separate re-ranking stage .

2.6.3 Discriminative Distortion

The standard distortion cost model used in phrase-based machine translation systems has two problems. First, it does not estimate the future cost of known required moves, thus increasing search errors. Second, the model penalizes distortion linearly, even when appropriate re-orderings are performed. To address these problems, Green et al. (2010) uses the Phrasal feature API to design a new discriminative distortion model that predicts word movement during translation and that estimates future cost. These extensions allow Green et al. (2010) to triple the distortion limit and provide a statistically significant improvement over the baseline.

2.6.4 Discriminative Reordering with Chinese Grammatical Relations

During translation, a source sentence can be more accurately reordered if the system knows something about the syntactic relationship between the words in the phrases being reordered. Chang et al. (2009) uses Phrasal to develop a discriminative reordering feature that examines the grammatical relationships be-

tween words in a Chinese source sentence and then scores the appropriateness of reordering their target side translations.

2.7 Conclusion

The Phrasal machine translation package has not only facilitated this work, but also the research projects of many others. For other researchers, one of the system's most valuable contributions has been its feature engineering API. While the work presented here only makes use of the system's flexible MERT component and support for training to other learning algorithms, the feature engineering API will likely be critical for trying to get the most out of newer evaluation metrics and to take advantage of the larger feature sets that can be used with alternative learning algorithms. Developing new features should allow the decoding model to better capture information that is being measured by newer evaluation metrics that perform deeper linguistic analysis but that is not usefully represented within existing decoding features (e.g., verb argument structure). The next chapter uses Phrasal to explore new approaches to MERT optimization.

Chapter 3

Regularization and Search for Minimum Error Rate Training

While minimum error rate training (MERT) is the most widely used learning procedure for statistical machine translation models, only a limited amount of work has been done exploring using different approaches to optimization for MERT. Software implementations of MERT usually just blindly use one of the line search based approaches such as Powell's method or coordinate descent. In this chapter, I use Phrasal to explore the effectiveness of different optimization algorithms.

Three optimization strategies are contrasted. The first two are Powell's method and coordinate descent. A novel random search direction algorithm is also purposed. In the general case, such an algorithm would be expected to perform worse than a more advanced technique such as Powell's method. Random search directions are used here in an effort to get more value out of the line search algorithm used with MERT that is able to efficiently find the global minimum/maximum along the direction being searched.

MERT exactly fits an evaluation metric on a given piece of training data. This raises the possibility that over-fitting may occur with learning producing a solution that does very well on the training data, while taking a sizable performance hit on unseen evaluation data. This chapter also explores whether it is possible to find more general solutions by regularizing the objective and smoothing the scores assigned to adjacent points.

The results provided here show that using random search directions outperforms both Powell's method and coordinate descent. This suggests that choosing search directions at random better exploits MERT's unique global minimum line search method. The experiments on regularization show that generalization performance of the system can be improved by smoothing the objective function used during training. How-

ever, regularization seems to only help Powell's method and coordinate descent, as adding regularization to the random search approach does not result in any further gains.

The next section reviews the technical details of minimum error rate training. Section 3.3 details the extensions to the algorithm proposed here and the motivation behind them. Section 3.4 and 3.5 provide experimental results, followed by a discussion in section 3.6

3.1 Minimum Error Rate Training

Let $\mathbf{F}$ be a collection of foreign sentences to be translated, with individual sentences $\mathbf{f}_0$, $\mathbf{f}_1$, $\cdots$, $\mathbf{f}_n$. For each $\mathbf{f}_i$, the surface form of an individual candidate translation is given by $\mathbf{e}_i$ with hidden state $\mathbf{h}_i$ associated with the derivation of $\mathbf{e}_i$ from $\mathbf{f}_i$. Each $\mathbf{e}_i$ is drawn from $\mathscr{E}$, which represents all possible strings a translation system can produce. The $(\mathbf{e}_i, \mathbf{h}_i, \mathbf{f}_i)$ triples are converted into vectors of m feature functions by $\Psi : \mathscr{E} \times \mathscr{H} \times \mathscr{F} \rightarrow \mathbb{R}^m$ whose dot product with the weight vector $\mathbf{w}$ assigns a score to each triple. The idealized translation process then is to find the highest scoring pair $(\mathbf{e}_i, \mathbf{h}_i)$ for each $\mathbf{f}_i$, or rather $(\mathbf{e}_i, \mathbf{h}_i) = \text{argmax}_{(\mathbf{e}\in\mathscr{E}, \mathbf{h}\in\mathscr{H})} \mathbf{w} \cdot \Psi(\mathbf{e}, \mathbf{h}, \mathbf{f})$.

The aggregate argmax for the entire data set $\mathbf{F}$ is given by equation (3.1).[1] This gives $\mathbf{E}_\mathbf{w}$ which represents the set of translations selected by the model for data set $\mathbf{F}$ when parameterized by the weight vector $\mathbf{w}$. Let's assume we have an automated measure of translation quality ℓ that maps the collection of translations $\mathbf{E}_\mathbf{w}$ onto some real valued loss, $\ell : \mathscr{E}^n \rightarrow \mathbb{R}$. For instance, in the experiments that follow, the loss corresponds to 1 minus the BLEU score assigned to $\mathbf{E}_\mathbf{w}$ for a given collection of reference translations.

$$(\mathbf{E}_\mathbf{w}, \mathbf{H}_\mathbf{w}) = \underset{(\mathbf{E}\in\mathscr{E}^n, \mathbf{H}\in\mathscr{H}^n)}{\text{argmax}} \mathbf{w} \cdot \Psi(\mathbf{E}, \mathbf{H}, \mathbf{F}) \tag{3.1}$$

Using n-best lists produced by a decoder to approximate $\mathscr{E}^n$ and $\mathscr{H}^n$, MERT searches for the weight vector $\mathbf{w}^*$ that minimizes the loss ℓ. Letting $\tilde{\mathbf{E}}_\mathbf{w}$ denote the result of the translation argmax w.r.t. the approximate hypothesis space, the MERT search is then expressed by equation (3.2). Notice the objective function being optimized is equivalent to the loss assigned by the automatic measure of translation quality,

[1] Here, the translation of the entire data set is treated as a single structured prediction problem using the feature function vector $\Psi(\mathbf{E}, \mathbf{H}, \mathbf{F}) = \sum_i^n \Psi(\mathbf{e}_i, \mathbf{h}_i, \mathbf{f}_i)$

id	Translation	$\log(\mathbf{P_{TM}}(\mathbf{f}\|\mathbf{e}))$	$\log(\mathbf{P_{LM}}(\mathbf{e}))$	BLEU-2
$\mathbf{e}^1$	This is it	-1.2	-0.1	29.64
$\mathbf{e}^2$	This is small house	-0.2	-1.2	63.59
$\mathbf{e}^3$	This is miniscule building	-1.6	-0.9	31.79
$\mathbf{e}^4$	This is a small house	-0.1	-0.9	100.00
ref	This is a small house			

Table 3.1: **Translations with Model Feature and Evaluation Scores:** Four hypothetical translations and their corresponding log model scores from a translation model $P_{TM}(f|e)$ and a language model $P_{LM}(e)$, along with their BLEU-2 scores according to the given reference translation. The MERT error surface for these translations is given in figure 3.1.

i.e. $\mathscr{O}(\mathbf{w}) = \ell(\tilde{\mathbf{E}}_{\mathbf{w}})$.

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, \ell(\tilde{\mathbf{E}}_{\mathbf{w}}) \tag{3.2}$$

After performing the parameter search, the decoder is then re-run using the weights $\mathbf{w}^*$ to produce a new set of n-best lists, which are then concatenated with the prior n-best lists in order to obtain a better approximation of $\mathscr{E}^n$ and $\mathscr{H}^n$. The parameter search given in (3.2) can then be performed over the improved approximation. This process repeats until either no novel entries are produced for the combined n-best lists or the weights change by less than some ε across iterations.

Unlike the objective functions associated with other popular learning algorithms, the objective $\mathscr{O}$ is piecewise constant over its entire domain. That is, while small perturbations in the weights, $\mathbf{w}$, will change the score assigned by $\mathbf{w} \cdot \Psi(\mathbf{e}, \mathbf{h}, \mathbf{f})$ to each triple, $(\mathbf{e}, \mathbf{h}, \mathbf{f})$, such perturbations will generally not change the ranking between the pair selected by the argmax, $(\mathbf{e}^*, \mathbf{h}^*) = \operatorname{argmax} \mathbf{w} \cdot \Psi(\mathbf{e}, \mathbf{h}, \mathbf{f})$, and any given competing pair $(\mathbf{e}', \mathbf{h}')$. However, at certain critical points, the score assigned to some competing pair $(\mathbf{e}', \mathbf{h}')$ will exceed that assigned to the prior winner $(\mathbf{e}^*_{w_{old}}, \mathbf{h}^*_{w_{old}})$. At this point, the pair returned by $\operatorname{argmax} \mathbf{w} \cdot \Psi(\mathbf{e}, \mathbf{h}, \mathbf{f})$ will change and loss ℓ will be evaluated using the newly selected $\mathbf{e}'$.

This is illustrated in figure 3.1, which plots the MERT objective function for a simple model with two parameters, w_{tm} & w_{lm}, and for which the space of possible translations, $\mathscr{E}$, consists of the four sentences

given in table 3.1.[2] Here, the loss ℓ is defined as 1.0−BLEU-2($\mathbf{e}$). That is, ℓ is the difference between a perfect BLEU score and the BLEU score calculated for each translation using unigram and bi-gram counts.

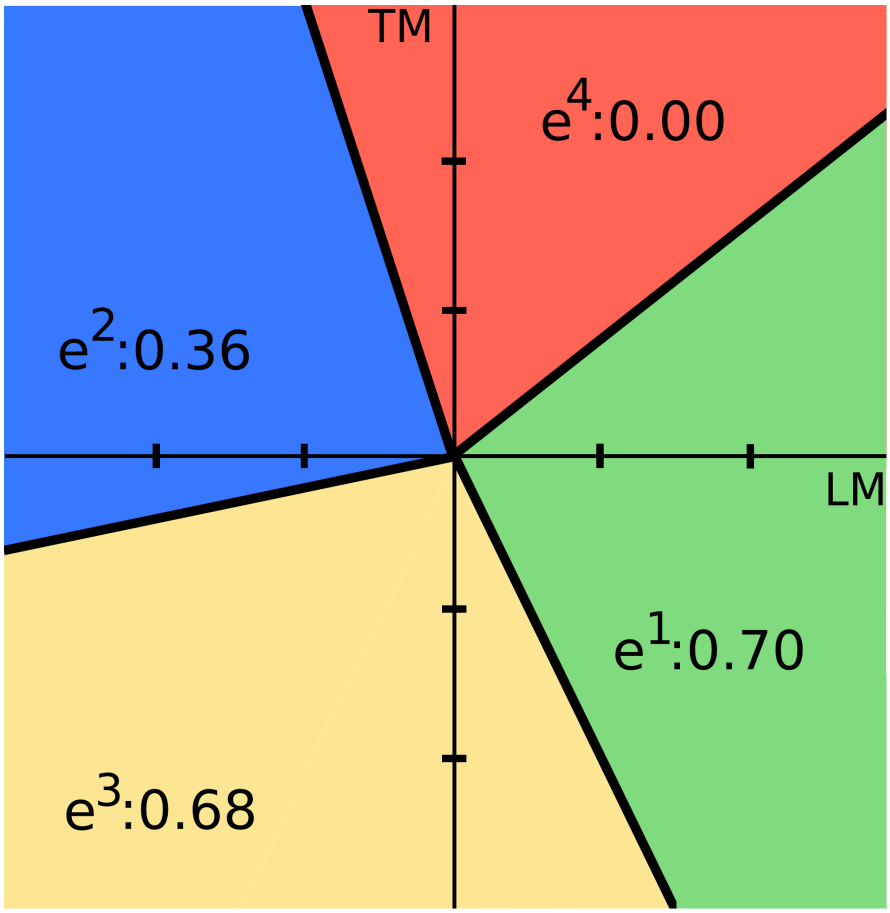

Figure 3.1: **MERT Objective Function:** Optimization surface resulting from the translation candidates and decoding model feature values given in table 3.1 being fit to the BLEU:2 evaluation metric. Regions are labeled with the translation that dominates within it, i.e. $\text{argmax}\,\mathbf{w}\cdot\Psi(\mathbf{e},\mathbf{f})$, and with their corresponding objective values, $1-\ell(\text{argmax}\,\mathbf{w}\cdot\Psi(\mathbf{e},\mathbf{f}))$.

The surface can be visualized as a collection of plateaus that all meet at the origin and then extend off into infinity. The latter property illustrates that the objective is scale invariant w.r.t. the weight vector $\mathbf{w}$. That is, since any vector $\mathbf{w}' = \lambda\mathbf{w}\ \forall_{\lambda>0}$ will still result in the same relative rankings of all possible translations according to $\mathbf{w}\cdot\Psi(\mathbf{e},\mathbf{h},\mathbf{f})$, such scaling will not change the translation selected by the argmax.

[2] For this example, we ignore the latent variables, $\mathbf{h}$, associated with the derivation of each $\mathbf{e}$ from the foreign sentence $\mathbf{f}$. If included, such variables would only change the graph in that multiple different derivations would be possible for each $\mathbf{e}^j$. If present, the graph could then include disjoint regions that all map to the same $\mathbf{e}^j$ and thus the same objective value.

Translation	m	b	1-best
e^1	-0.1	-1.25	(0.86,+∞]
e^2	-1.2	-0.8	(-0.83,0.88)
e^3	-0.9	-2.05	n/a
e^4	-0.9	-0.55	[−∞,-0.83]

Table 3.2: **Line Search Parameters:** Slopes, m, intercepts, b, and 1-best ranges for the 4 translations given in table 3.1 during a line search along the coordinate w_{lm}, with a starting point of $(w_{tm}, w_{lm}) = (1.0, 0.5)$. This line search is illustrated in figure 3.2.

At the boundaries between regions, the objective is undefined, as 2 or more candidates are assigned identical scores by the model. Thus, it is unclear what should be returned by the argmax for subsequent scoring by ℓ.

Since the objective is piecewise constant, it cannot be minimized using gradient descent or even the sub-gradient method. Two applicable methods include downhill simplex and Powell's method (Press et al., 2007). The former attempts to find a local minimum in an n dimensional space by iteratively shrinking or growing an $n+1$ vertex simplex[3] based on the objective values of the current vertex points and select nearby points. In contrast, Powell's method operates by starting with a single point in weight space, and then performing a series of line minimizations until no more progress can be made. In this chapter, I focus on line minimization based techniques, such as Powell's method.

3.1.1 Global Minimum Along a Line

Even without gradient information, numerous methods can be used to find, or approximately find, local minima along a line. However, by exploiting the fact that the underlying scores assigned to competing hypotheses, $\mathbf{w} \cdot \Psi(\mathbf{e}, \mathbf{h}, \mathbf{f})$, vary linearly w.r.t. changes in the weight vector, $\mathbf{w}$, Och (2003) proposed a strategy for finding the global minimum along any given search direction.

The insight behind the algorithm is as follows. Let's assume we are examining two competing translation/derivation pairs, $(\mathbf{e}^1, \mathbf{h}^1)$ & $(\mathbf{e}^2, \mathbf{h}^2)$. Further, let's say the score assigned by the model to $(\mathbf{e}^1, \mathbf{h}^1)$ is greater than $(\mathbf{e}^2, \mathbf{h}^2)$, i.e. $\mathbf{w} \cdot \Psi(\mathbf{e}^1, \mathbf{h}^1, \mathbf{f}) > \mathbf{w} \cdot \Psi(\mathbf{e}^2, \mathbf{h}^2, \mathbf{f})$. Since the scores of the two vary linearly along any search direction, $\mathbf{d}$, we can find the point at which the model's relative preference for

[3] A simplex can be thought of as a generalization of a triangle to arbitrary dimensional spaces.

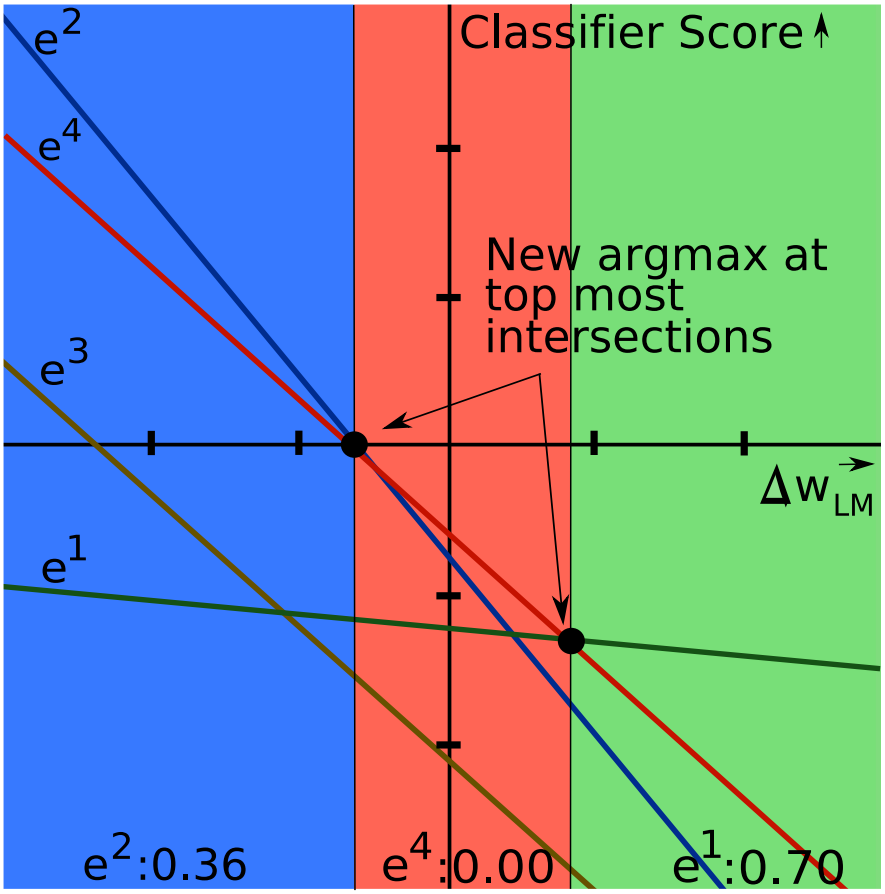

Figure 3.2: **Global Minimum Line Search:** Illustration of how the model score assigned to each candidate translation varies during a line search along the coordinate direction w_{lm} with a starting point of $(w_{tm}, w_{lm}) = (1.0, 0.5)$. Each plotted line corresponds to the model score for one of the translation candidates. The vertical bands are labeled with the hypothesis that dominates in that region. The transitions between bands result from the dotted intersections between 1-best lines.

the competing pairs switches as $p = \frac{\mathbf{w}\cdot\Psi(\mathbf{e}^1,\mathbf{h}^1,\mathbf{f}) - \mathbf{w}\cdot\Psi(\mathbf{e}^2,\mathbf{h}^2,\mathbf{f})}{\mathbf{d}\cdot\Psi(\mathbf{e}^2,\mathbf{h}^2,\mathbf{f}) - \mathbf{d}\cdot\Psi(\mathbf{e}^1,\mathbf{h}^1,\mathbf{f})}$. At this particular point, we have the equality $(p\mathbf{d}+\mathbf{w})\cdot\Psi(\mathbf{e}^1,\mathbf{h}^1,\mathbf{f}) = (p\mathbf{d}+\mathbf{w})\cdot\Psi(\mathbf{e}^2,\mathbf{h}^2,\mathbf{f})$, or rather the point at which the scores assigned by the model to the candidates intersect along search direction $\mathbf{d}$.[4] Such points correspond to the boundaries between adjacent plateaus in the objective, as prior to the boundary the loss function ℓ is computed using the translation, $\mathbf{e}^1$, and after the boundary it is computed using $\mathbf{e}^2$.

To find the global minimum for a search direction $\mathbf{d}$, we move along $\mathbf{d}$ and for each plateau we identify all the points at which the score assigned by the model to the current 1-best translation intersects the

[4] Notice that, this point only exists if the slopes of the candidates' model scores along $\mathbf{d}$ are not equivalent, i.e. if $\mathbf{d}\cdot\Psi(\mathbf{e}^2,\mathbf{h}^2,\mathbf{f}) \neq \mathbf{d}\cdot\Psi(\mathbf{e}^1,\mathbf{h}^1,\mathbf{f})$.

score assigned to competing translations. At the closest such intersection, we have a new 1-best translation. Moving to the plateau associated with this new 1-best, we then repeat the search for the nearest subsequent intersection. This continues until we know what the 1-best translations are for all points along **d**. The global minimum can then be found by examining ℓ once for each of these.

Let's return briefly to the earlier example given in table 3.1. Starting at position $(w_{tm}, w_{lm}) = (1.0, 0.5)$ and searching along the w_{lm} coordinate, i.e. $(d_{tm}, d_{lm}) = (0.0, 1.0)$, table 3.2 gives the line search slopes, $m = \mathbf{d} \cdot \Psi(\mathbf{e}, \mathbf{h}, \mathbf{f})$, and intercepts, $b = \mathbf{w} \cdot \Psi(\mathbf{e}, \mathbf{h}, \mathbf{f})$, for each of the four candidate translations. Using the procedure just described, we can then find what range of values along **d** each candidate translation is assigned the highest relative model score. Figure 3.2 illustrates how the score assigned by the model to each of the translations changes as we move along **d**. Each of the banded regions corresponds to a plateau in the objective, and each of the top most line intersections represents the transition from one plateau to the next. Note that, while the surface that is defined by the line segments with the highest classifier score for each region is convex, this is not a convex optimization problem as we are optimizing over the loss ℓ rather than classifier score. Since the loss ℓ is determined by running an external evaluation metric over the best scoring hypothesis within each region, it can exhibit arbitrary variations along the line being searched.

Pseudo-code for the line search is given in algorithm 2. Letting n denote the number of foreign sentences, **f**, in a dataset, and having m denote the size of the individual n-best lists, $|l|$, the time complexity of the algorithm is given by $O(nm^2)$. This is seen in that each time we check for the nearest intersection to the current 1-best for some n-best list l, we must calculate its intersection with all other candidate translations that have yet to be selected as the 1-best. And, for each of the n n-best lists, this may have to be done up to $m-1$ times.

3.1.2 Search Strategies

In this section, I review two search strategies that, in conjunction with the line search just described, can be used to drive MERT. The first, Powell's method, was advocated by Och (2003) when MERT was first

Algorithm 2 MERT Line Search: Och 2003's line search method to find the global minimum in the loss, ℓ, when starting at the point $\mathbf{w}$ and searching along the direction $\mathbf{d}$ using the candidate translations given in the collection of n-best lists $\mathscr{L}$.

Input: $\mathscr{L}$, $\mathbf{w}$, $\mathbf{d}$, ℓ
$\mathscr{I} \Leftarrow \{\}$
for $l \in \mathscr{L}$ **do**
  for $e \in l$ **do**
    m{e} $\Leftarrow$ e.features $\cdot$ $\mathbf{d}$
    b{e} $\Leftarrow$ e.features $\cdot$ $\mathbf{w}$
  end for
  $\text{best}_n \Leftarrow \text{argmax}_{e \in l}$ m{e} {b{e} breaks ties}
  loop
    $\text{best}_{n+1} = \text{argmin}_{e \in l} \max\left(0, \frac{b\{best_n\} - b\{e\}}{m\{e\} - m\{best_n\}}\right)$
    intercept $\Leftarrow \max\left(0, \frac{b\{best_n\} - b\{best_{n+1}\}}{m\{best_{n+1}\} - m\{best_n\}}\right)$
    if intercept > 0 **then**
      add($\mathscr{I}$, intercept)
    else
      break
    end if
  end loop
end for
add($\mathscr{I}$, $\max(\mathscr{I}) + 2\varepsilon$)
$i_{best} = \text{argmin}_{i \in \mathscr{I}} \text{argmax}_{\ell}(\mathscr{L}, \mathbf{w} + (i - \varepsilon) \cdot \mathbf{d})$
return $\mathbf{w} + (i_{best} - \varepsilon) \cdot \mathbf{d}$

introduced for statistical machine translation. The second, which I call Koehn-coordinate descent (KCD),[5] is used by the MERT utility packaged with the popular Moses statistical machine translation system (Koehn et al., 2007).

3.1.2.1 Powell's Method

Powell's method (Press et al., 2007) attempts to efficiently search the objective by constructing a set of mutually non-interfering search directions. The basic procedure is as follows: (i) A collection of search directions is initialized to be the coordinates of the space being searched; (ii) The objective is minimized by looping through the search directions and performing a line minimization for each; (iii) A new search direction is constructed that summarizes the cumulative direction of the progress made during step (ii) (i.e.,

[5] Moses uses David Chiang's CMERT package. Within the source file mert.c, the function that implements the overall search strategy, optimize_koehn(), is based on Philipp Koehn's Perl script for MERT optimization that was distributed with Pharaoh.

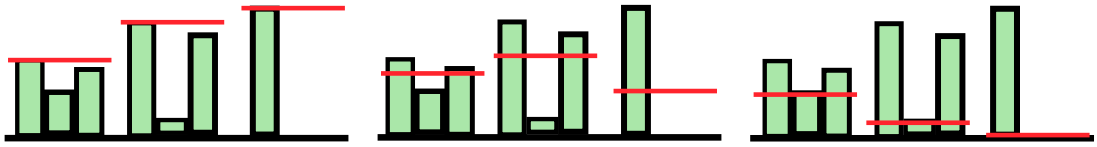

Figure 3.3: **Regularization Heuristics:** Line search based smoothing heuristics, from left to right: (i) the maximum loss of adjacent plateaus, (ii) the average loss of adjacent plateaus, and (iii) no regularization. Each set of bars represents adjacent plateaus along the line being searched, with the height of the bars representing their associated loss. The vertical lines indicate the surrogate loss values used for the center region under each of the schemes (i-iii).

$\mathbf{d}_{new} = \mathbf{w}_{pre_{ii}} - \mathbf{w}_{post_{ii}}$). After a line minimization is performed along $\mathbf{d}_{new}$, it is used to replace one of the existing search directions. (iv) The process repeats until no more progress can be made. For a quadratic function of n variables, this procedure comes with the guarantee that it will reach the minimum within n iterations of the outer loop. However, since Powell's method is usually applied to non-quadratic optimization problems, a typical implementation will forgo the quadratic convergence guarantees in favor of a heuristic scheme that allows for better navigation of complex surfaces.

3.1.2.2 Koehn's Coordinate Descent

KCD is a variant of coordinate descent that, at each iteration, moves along the coordinate which allows for the most progress in the objective. In order to determine which coordinate this is, the routine performs a trial line minimization along each. It then updates the weight vector with the one that it found to be most successful. While much less sophisticated that Powell, the results indicate that this method may be marginally more effective at optimizing the MERT objective.[6]

3.2 Extensions

In this section, I present and motivate two novel extensions to MERT. The first is a stochastic alternative to the Powell and KCD search strategies, while the second is an efficient method for regularizing the objective.

[6] While I am not aware of any previously published results that demonstrate this, it is likely that I were not the first to make this discovery as even though Moses' MERT implementation includes a vestigial implementation of Powell's method, the code is hardwired to call optimize_koehn() rather than the routine for Powell.

3.2.1 Random Search Directions

One significant advantage of Powell's algorithm over coordinate descent is that it can optimize along diagonal search directions in weight space. That is, given a model with a dozen or so features, it can explore gains that are to be had by simultaneously varying two or more of the feature weights. In general, the diagonals that Powell's method constructs allow it to walk objective functions more efficiently than coordinate descent (Press et al., 2007). However, given that I have a line search algorithm that will find the global minima along any given search direction, diagonal search may be of even more value. That is, similar to ridge phenomenon that arise in traditional hill climbing search, it is possible that there are points in the objective that are the global minimum along any given coordinate direction, but are not the global minimum along diagonal directions.

However, one substantial disadvantage for Powell is that the assumptions it uses to build up the diagonal search directions do not hold in the present context. Specifically, the search directions are built up under the assumption that near a minimum the surface looks approximately quadratic and that we are performing local line minimizations within such regions. However, since we are performing global line minimizations, it is possible for the algorithm to jump from the region around one local minimum to another. If Powell's method has already started to tune its search directions for the prior minima, it will likely be less effective in its efforts to search the new region. To this extent, coordinate descent will be more robust than Powell as it has no assumptions that are violated when such a jump occurs.

One way of salvaging Powell's algorithm in this context would be to incorporate additional heuristics that detect when the algorithm has jumped from the region around one local minimum to another. When this occurs, the search directions could be reset to the coordinates of the space. However, I opt for a simpler solution, which like Powell's algorithm performs searches along diagonals in the space, but that like coordinate descent is sufficiently simple that the algorithm will not be confused by sudden jumps between regions.

Specifically, the search procedure chooses directions at random such that each component is distributed according to a Gaussian,[7] $\mathbf{d}$ s.t. $d_i \sim N(0,1)$. This allows the procedure to minimize along diagonal search directions, while making essentially no assumptions regarding the characteristics of the objective or

[7] However, I speculate that similar results could be obtained using a uniform distribution over $(-1,1)$

the relationship between a series of sequential line minimizations. In the results that follow, I show that, perhaps surprisingly, this simple procedure outperforms both KCD and Powell's method.

3.2.2 Regularization

One potential drawback of MERT, as it is typically implemented, is that it attempts to find the best possible set of parameters for a training set without making any explicit efforts to find a set of parameters that can be expected to generalize well. For example, let's say that for some objective there is a very deep but narrow minimum that is surrounded on all sides by very bad objective values. That is, the BLEU score at the minima might be 39.1 while all surrounding plateaus have a BLEU score that is < 10. Intuitively, such a minimum would be a very bad solution, as the resulting parameters would likely exhibit very poor generalization to other data sets. This could be avoided by regularizing the surface in order to eliminate such spurious minima.

One candidate for performing such regularization is the continuous approximation of the MERT objective, $\mathscr{O} = \mathbb{E}_{p_{\mathbf{w}}}(\ell)$. Och (2003) claimed that this approximation achieved essentially equivalent performance to that obtained when directly using the loss as the objective, $\mathscr{O} = \ell$. However, Zens et al. (2007) found that $\mathscr{O} = \mathbb{E}_{p_{\mathbf{w}}}(\ell)$ achieved substantially better test set performance than $\mathscr{O} = \ell$, even though it performs slightly worse on the data used to train the parameters. Similarly, Smith and Eisner (2006) reported test set gains for the related technique of minimum risk annealing, which incorporates a temperature parameter that trades off between the smoothness of the objective and the degree it reflects the underlying piecewise constant error surface. However, the most straightforward implementation of such methods requires a loss that can be applied at the sentence level. If the evaluation metric of interest does not have this property (e.g. BLEU), the loss must be approximated using some surrogate, with successful learning then being tied to how well the surrogate captures the critical properties of the underlying loss.

The techniques of Zens et al. (2007) & Smith and Eisner (2006) regularize by implicitly smoothing over nearby plateaus in the error surface. I propose an alternative scheme that operates directly on the piecewise constant objective and that mitigates the problem of spurious local minima by explicitly smoothing over adjacent plateaus during the line search. That is, when assessing the desirability of any given plateau,

I examine a fixed window *w* of adjacent plateaus along the direction being searched and combine their evaluation scores. I explore two combination methods, *max* and *average*. The former, *max*, assigns each plateau an objective value that is equal to the maximum objective value in its surrounding window, while *average* assigns a plateau an objective value that is equal to its window's average. Figure 3.3 illustrates both methods for regularizing the plateaus and contrasts them with the case where no regularization is used. Notice that, while both methods discount spurious pits in the objective, *average* still does place some value on isolated deep plateaus, and *max* discounts them completely.

Note that one potential weakness of this scheme is the value assigned by the regularized objective to any given point differs depending on the direction being searched. As such, it has the potential to wreak havoc on methods such as Powell's, which effectively attempt to learn about the curvature of the objective from a sequence of line minimizations.

3.3 Experiments

Three sets of experiments were performed. For the first set, I compare the performance of Powell's method, KCD, and my novel stochastic search strategy. I then evaluate the performance of all three methods when the objective is regularized using the average of adjacent plateaus for window sizes varying from 3 to 7. Finally, I repeat the regularization experiment, but using the maximum objective value from the adjacent plateaus. These experiments were performed using the Chinese English evaluation data provided for NIST MT eval 2002, 2003, and 2005. MT02 was used as a dev set for MERT learning, while MT03 and MT05 were used as the test sets.

For all experiments, MERT was performed using *n*-best lists from the decoder of size 100. During each iteration, the MERT search was performed once with a starting point of the weights used to generate the most recent set of *n*-best lists and then 5 more times using randomly selected starting points.[8] Of these, I retain the weights from the search that obtained the lowest objective value. Training continued until either decoding produced no novel entries for the combined *n*-best lists or none of the parameter values changed

[8] Only 5 random restarts were used due to time constraints. Ideally, a sizable number of random restarts should be used in order to minimize the degree to which the results are influenced by some runs receiving starting points that are better in general or perhaps better/worse w.r.t. their specific optimization strategy.

Method	Dev MT02	Test MT03	Test MT05
KCD	30.967	30.778	29.580
Powell	30.638	30.692	29.780
Random	31.681	31.754	30.191

Table 3.3: **Optimization Search Strategy Performance:** BLEU scores obtained by models trained using three different parameter search strategies: Powell's method, KCD, and stochastic search.

by more than 1e-5 across subsequent iterations.

3.3.1 System

Experiments were run using the Phrasal machine translation system (Cer et al., 2010b). I made use of a stack size of 50, as this allowed for faster experiments while only performing modestly worse than a stack of 200. The distortion limit was set to 6. The decoder was configured to retrieve 20 translation options for each unique source phrase.

The phrase-table was built using $1,140,693$ sentence pairs sampled from the GALE Y2 training data. The Chinese data was word segmented using the GALE Y2 retest release of the Stanford CRF segmenter (Tseng et al., 2005). Phrases were extracted using the typical approach described in Koehn et al. (2003) of running GIZA++ (Och & Ney, 2003) in both directions and then merging the alignments using the grow-diag-final heuristic. From the merged alignments, a bi-directional lexical reordering model conditioned on the source and the target phrases (Tillmann, 2004; Koehn et al., 2007) was also extracted. A 5-gram language model was created using the SRI language modeling toolkit (Stolcke, 2002) and trained using the Gigaword corpus and English sentences from the parallel data.

3.4 Results

As illustrated in table 3.3, Powell's method and KCD achieve a very similar level of performance, with KCD modestly outperforming Powell on the MT03 test set while Powell modestly outperforms coordinate descent on the MT05 test set. Moreover, the fact that Powell's algorithm did not perform better than KCD on

Method	Win Avg	Dev MT02	Test MT03	Test MT05	Method	Win Max	Dev MT02	Test MT03	Test MT05
Coord.	none	30.967	30.778	29.580	Coord.	none	30.967	30.778	29.580
	3	31.665	**31.675**	**30.266**		3	31.536	**31.927**	**30.334**
	5	31.317	31.229	**30.182**		5	31.484	**31.702**	29.687
	7	31.205	**31.824**	30.149		7	31.627	31.294	**30.199**
Powell	none	30.638	30.692	29.780	Powell	none	30.638	30.692	29.780
	3	31.333	31.412	29.890		3	31.428	30.944	29.598
	5	31.748	**31.777**	**30.334**		5	31.407	**31.596**	30.090
	7	31.249	**31.571**	30.161		7	30.870	30.911	29.620
Random	none	31.681	31.754	30.191	Random	none	31.681	31.754	30.191
	3	31.548	31.778	30.263		3	31.179	30.898	29.529
	5	31.336	31.647	30.415		5	30.903	31.666	29.963
	7	30.501	29.336	28.372		7	31.920	31.906	30.674

Table 3.4: **Effectiveness of Smoothing Heuristics:** BLEU scores obtained when regularizing using the average loss of adjacent plateaus, left, and the maximum loss of adjacent plateaus, right. The *none* entry for each search strategy represents the baseline where no regularization is used. Statistically significant test set gains, $p < 0.01$, over the respective baselines are in bold face.

the training data,[9] and in fact actually performed modestly worse, suggests that Powell's additional search machinery does not provide much benefit for MERT objectives.

Similarly, the fact that the stochastic search obtains a much higher dev set score than either Powell or KCD indicates that it is doing a better job of optimizing the objective than either of the two alternatives. These gains suggest that stochastic search does make better use of the global minimum line search than the alternative methods. Or, alternatively, it strengthens the claim that the method succeeds at combining one of the critical strengths of Powell's method, diagonal search, with coordinate descent's robustness to the sudden jumps between regions that result from global line minimization. Using an approximate randomization test for statistical significance (Riezler & Maxwell, 2005), and with KCD as a baseline, the gains obtained by stochastic search on MT03 are statistically significant ($p = 0.002$), as are the gains on MT05 ($p = 0.005$).

Table 3.4 indicates that performing regularization by either averaging or taking the maximum of adjacent plateaus during the line search leads to gains for both Powell's method and KCD. However, no reliable additional gains appear to be had when stochastic search is combined with regularization.

It may seem surprising that the regularization gains for Powell & KCD are seen not only in the test

[9] This indicates that Powell failed to find a deeper minima in the objective, since recall that the unregularized objective is equivalent to the model's dev set performance.

sets but on the dev set as well. That is, in typical applications, regularization slightly decreases performance on the data used to train the model. However, this trend can in part be accounted for by the fact that during training, MERT is using *n*-best lists for objective evaluations rather than the more expensive process of running the decoder for each point that needs to be checked. As such, during each iteration of training, the decoding performance of the model actually represents its generalization performance relative to what was learned from the *n*-best lists created during prior iterations. Moreover, better generalization from the prior *n*-best lists can also help drive subsequent learning as there will then be more high quality translations on the *n*-best lists used for future iterations of learning. Additionally, regularization can reduce search errors by reducing the risk of getting stuck in spurious low loss pits that are in otherwise bad regions of the space.

3.5 Conclusions

In this chapter, I have presented two methods for improving the performance of MERT. The first is a novel stochastic search strategy that appears to make better use of Och (2003)'s algorithm for finding the global minimum along any given search direction than either coordinate descent or Powell's method. The second is a simple regularization scheme that leads to performance gains for both coordinate descent and Powell's method. However, no further gains are obtained by combining the stochastic search with regularization of the objective.

While it is useful to be able to efficiently maximize a system's performance on a chosen evaluation metric, it is perhaps even more critical that the evaluation metric being optimized improves the actual quality of the translations produced by the system rather than just the evaluation score. The next chapter examines the relationship between the evaluation metric optimized and the actual quality of the resulting machine translation systems. This is done by training systems to a variety of different popular evaluation metrics and then having the translations produced by the systems evaluated by human judges. The chapter also examines the stability of training to the different evaluation metrics, and the performance of systems on evaluation metrics other than the metric that was used to train them.

Chapter 4

The Best Lexical Metric for Phrase-Based Statistical Machine Translation System Optimization

4.1 Introduction

In the last chapter, all of the machine translation models were tuned to optimize the BLEU score. This, of course, is not an accident since up until very recently nearly all translation systems were trained to optimize BLEU. More recently an increasing number of systems have been trained to the newer TER metric, since a human edited variant of the metric, hTER, is being used as the official evaluation metric for the DARPA GALE[1] project. Even so, most research results are presented using systems tuned to the BLEU metric. This is done since the machine translation community has come to expect experimental results presented in terms of BLEU. For any given standard evaluation set, publishing BLEU scores makes it easy to gauge to quality of a system and the relative significance of the experimental results. To do well on the BLEU score, it then makes sense to tune the system to BLEU.

However, the development of improved evaluation metrics is a very active area of research. A number of newer metrics correlate better with human judgments than BLEU. It is reasonable to wonder if tuning to these metrics would result in better systems that produce translations that are preferred by humans to those produced by a system trained to a simpler metric like BLEU. This chapter explores how optimizing toward various automatic evaluation metrics (BLEU, METEOR, NIST, TER, and TERp) affects the resulting model. A state-of-the-art machine translation system is trained using MERT on many parameterizations of each metric. The resulting models are evaluated using the other metrics and also by human judges.

[1] Defense Advanced Research Projects Agency: Global Autonomous Language Exploitation

Since their introduction, automated measures of machine translation quality have played a critical role in the development and evolution of translation systems. While such metrics were initially intended for evaluation, popular training methods such as minimum error rate training (MERT) (Och, 2003) and margin infused relaxed algorithm (MIRA) (Crammer & Singer, 2003; Watanabe et al., 2007; Chiang et al., 2008; Chiang et al., 2009) can be used to train translation models toward a specific evaluation metric. This makes the quality of the resulting model dependent on how accurately the automatic metric actually reflects human preferences.

For the reasons given above, the most popular metric for both comparing systems and tuning decoding models has been BLEU. While BLEU (Papineni et al., 2002) is relatively simple, scoring translations according to their *n*-gram overlap with reference translations, it still achieves a reasonable correlation with human judgments of translation quality. It is also robust enough to use for automatic optimization. However, BLEU does have a number of shortcomings. It does not penalize *n*-gram scrambling (Callison-Burch et al., 2006), and since it is not aware of synonymous words or phrases, it can inappropriately penalize translations that use them.

Recently, there have been efforts to develop better evaluation metrics. Metrics such as *Translation Edit Rate* (TER) (Snover et al., 2006; Snover et al., 2009) and METEOR[2] (Lavie & Denkowski, 2009) perform a more sophisticated analysis of the translations being evaluated and the scores they produce tend to achieve a better correlation with human judgments than those produced by BLEU (Snover et al., 2009; Lavie & Denkowski, 2009; Przybocki et al., 2008; Snover et al., 2006).

These better correlations suggest that one might obtain higher quality translations by making use of the newer metrics when training translation models. It is expected that training on a specific metric will produce the best performing model according to that metric. Doing better on metrics that better reflect human judgments seems to imply the translations produced by the model would be preferred by human judges.

However, there are four potential problems. First, some metrics could be susceptible to systematic exploitation by the training algorithm and result in model translations that have a high score according to the

[2] METEOR: Metric for Evaluation of Translation with Explicit ORdering.

evaluation metric but that are of low quality.[3] Second, other metrics may result in objective functions that are harder to optimize. Third, some may result in better generalization performance than others at test time by not encouraging overfitting of the training data. Finally, as a practical concern, metrics used for training cannot be too slow.

This chapter systematically explores these four issues for the most popular metrics available to the machine translation community. It examines how well models perform both on the metrics on which they were trained and on the other alternative metrics. Multiple models are trained using each metric in order to determine the stability of the resulting models. Select models are scored by human judges in order to determine how performance differences obtained by tuning to different automated metrics relate to actual human preferences. The next section reviews the operation of the metrics. This is followed with two sets of core results: machine evaluation in section 4.4, and human evaluation in section 4.5.

4.2 Evaluation Metrics

Designing good automated metrics for evaluating machine translations is challenging due to the variety of acceptable translations for each foreign sentence. Popular metrics produce scores primarily based on matching sequences of words in the system translation to those in one or more reference translations. The metrics primarily differ in how they account for reorderings, synonyms, and local paraphrases.

4.2.1 BLEU

BLEU (Papineni et al., 2002) uses the percentage of n-grams found in machine translations that also occur in the reference translations. These n-gram precisions are calculated separately for different n-gram lengths and then combined using a geometric mean. The score is then scaled by a brevity penalty if the candidate translations are shorter than the references, $\text{BP} = \min(1.0, e^{1-\text{len}(R)/\text{len}(T)})$. Equation 4.1 gives BLEU using n-grams up to length N for a corpus of candidate translations T and reference translations R. A variant of BLEU called the NIST metric (Doddington, 2002) weights n-gram matches by how informative

[3] For example, BLEU computed without the brevity penalty would likely result in models that have a strong preference for generating pathologically short translations.

they are.

$$\text{BLEU:}N = \left(\prod_{n=1}^{N} \frac{n\text{-grams}(T \bigcap R)}{n\text{-grams}(T)} \right)^{\frac{1}{N}} \text{BP} \tag{4.1}$$

While easy to compute, BLEU has a number of shortcomings. Since the order of matching n-grams is ignored, n-grams in a translation can be randomly rearranged around non-matching material or other n-gram breaks without harming the score. BLEU also does not explicitly check whether information is missing from the candidate translations, as it only examines what fraction of candidate translation n-grams are in the references and not what fraction of references n-grams are in the candidates (i.e., BLEU ignores n-gram recall). Finally, the metric does not account for words and phrases that have similar meanings.

4.2.2 METEOR

METEOR (Lavie & Denkowski, 2009) computes a one-to-one alignment between matching words in a candidate translation and a reference. If a word matches multiple other words, preference is given to the alignment that reorders the words the least, with the amount of reordering measured by the number of crossing alignments. Alignments are first generated for exact matches between words. Additional alignments are created by repeatedly running the alignment procedure over unaligned words, first allowing for matches between word stems, and then allowing matches between words listed as synonyms in WordNet. From the final alignment, the candidate translation's unigram precision and recall is calculated, $P = \frac{\text{matches}}{\text{length trans}}$ and $R = \frac{\text{matches}}{\text{length ref}}$. These two are then combined into a weighted harmonic mean (4.2). To penalize reorderings, this value is then scaled by a fragmentation penalty based on the number of chunks the two sentences would need to be broken into to allow them to be rearranged with no crossing alignments, $P_{\beta,\gamma} = 1 - \gamma\left(\frac{\text{chunks}}{\text{matches}}\right)^{\beta}$.

$$F_\alpha = \frac{PR}{\alpha P + (1-\alpha)R} \tag{4.2}$$

$$\text{METEOR}_{\alpha,\beta,\gamma} = F_\alpha \cdot P_{\beta,\gamma} \tag{4.3}$$

Equation 4.3 gives the final METEOR score as the product of the unigram harmonic mean, F_α, and the fragmentation penalty, $P_{\beta,\gamma}$. The free parameters α, β, and γ can be used to tune the metric to human judgments on a specific language and variation of the evaluation task (e.g., ranking candidate translations vs. reproducing judgments of translations adequacy and fluency).

4.2.3 Translation Edit Rate

TER (Snover et al., 2006) searches for the shortest sequence of edit operations needed to turn a candidate translation into one of the reference translations. The allowable edits are the insertion, deletion, and substitution of individual words as well as swaps of adjacent sequences of words. The swap operation differentiates TER from the simpler word error rate (WER) metric (Nießen et al., 2000), which only makes use of insertions, deletions, and substitutions. Swaps prevent phrase reorderings from being excessively penalized. Once the shortest sequence of operations is found,[4] TER is calculated simply as the number of required edits divided by the reference translation length, or average reference translation length when multiple references are available (4.4).

$$\text{TER} = \frac{\text{min edits}}{\text{avg ref length}} \tag{4.4}$$

TER-Plus (TERp) (Snover et al., 2009) extends TER by allowing the cost of edit operations to be tuned in order to maximize the metric's agreement with human judgments. TERp also introduces three new edit operations: word stem matches, WordNet synonym matches, and multi-word matches using a table of scored paraphrases.

4.3 Experiments

Experiments were run using the Phrasal machine translation package (Cer et al., 2010b). During decoding, I made use of a stack size of 100, set the distortion limit to 6, and retrieved 20 translation options for each unique source phrase.

[4] Since swaps prevent TER from being calculated exactly using dynamic programming, a beam search is used and this can overestimate the number of required edits.

Using the selected metrics, I train both Chinese to English and Arabic to English models.[5] The Chinese to English models are trained using NIST MT02 and evaluated on NIST MT03. The Arabic to English experiments use NIST MT06 for training and GALE dev07 for evaluation. The resulting models are scored using all of the stand alone metrics used during training.

4.3.1 Arabic to English

The Arabic to English system was based on Stanford's well ranking 2009 NIST submission (Galley et al., 2009). The phrase-table was extracted using all of the allowed resources for the constrained Arabic to English track. Word alignment was performed using the Berkeley cross-EM aligner (Liang et al., 2006b). Phrases were extracted using the grow heuristic (Koehn et al., 2003). However, all phrases that have a $P(e|f) < 0.0001$ were thrown away in order to reduce the size of the phrase-table. From the aligned data, a hierarchical reordering model was extracted that is similar to popular lexical reordering models (Koehn et al., 2007) but that models swaps containing more than just one phrase (Galley & Manning, 2008). A 5-gram language model was created with the SRI language modeling toolkit (Stolcke, 2002) using all of the English material from the parallel data employed to train the phrase-table as well as Xinhua Chinese English Parallel News (LDC2002E18).[6] The resulting decoding model has 16 features that are optimized during MERT.

4.3.2 Chinese to English

For the Chinese to English system, the phrase-table was built using 1,140,693 sentence pairs sampled from the GALE Y2 training data. The Chinese sentences were word segmented using the 2008 version of Stanford Chinese Word Segmenter (Chang et al., 2008; Tseng et al., 2005). Phrases were extracted by running GIZA++ (Och & Ney, 2003) in both directions and then merging the alignments using the grow-diag-final heuristic (Koehn et al., 2003). From the merged alignments, a bidirectional lexical reordering model conditioned on the source and the target phrases was also extracted (Koehn et al., 2007). A 5-gram language model was created with the SRI language modeling toolkit (Stolcke, 2002) and trained using the

[5] Given the amount of time required to train a TERpA model, I only present TERpA results for Chinese to English.

[6] In order to run multiple experiments in parallel on the computers available to me, the system I use for this work differs from the Stanford NIST 2009 submission in that I remove the Google *n*-gram language model. This results in a performance drop of less than 1.0 BLEU point on the dev data.

Gigaword corpus and English sentences from the parallel data. The resulting decoding model has 14 features to be trained.

4.4 Results

As seen in tables 4.2 and 4.4, the evaluation metric I use during training has a substantial impact on performance as measured by the various other metrics. There is a clear block structure where the best class of metrics to train on is the same class that is used during evaluation. Within this block structure, I make three primary observations. First, the best performing model according to any specific metric *configuration* is usually not the model I trained to that configuration. In the Chinese results, the model trained on BLEU:3 scores 0.74 points higher on BLEU:4 than the model actually trained to BLEU:4. In fact, the BLEU:3 trained model outperforms all other models on BLEU:*N* metrics. For the Arabic results, training on NIST scores 0.27 points higher on BLEU:4 than training on BLEU:4, and outperforms all other models on BLEU:*N* metrics.

Second, the edit distance based metrics (WER, TER, TERp, TERpA)[7] seem to be nearly interchangeable. While the introduction of swaps allows the scores produced by the TER metrics to achieve better correlation with human judgments, the models are apparently unable to exploit this during training. This might be due to the monotone nature of the reference translations and the fact that having multiple references reduces the need for reorderings. However, it is possible that differences between training to WER and TER would become more apparent using models that allow for longer distance reorderings or that do a better job of capturing what reorderings are acceptable.

Third, with the exception of BLEU:1, the performance of the BLEU, NIST, and the METEOR α=.5 models appears to be more robust across the other evaluation metrics than the standard METEOR, METEOR ranking, and edit distance based models (WER, TER, TERp, an TERpA). The latter models tend to do quite well on metrics similar to what they were trained on, while performing particularly poorly on the other metrics. For example, on Chinese, the TER and WER models perform very well on other edit distance based

[7] In my implementation of multi-reference WER, I use the length of the references that result in the lowest sentence level WER to divide the edit costs. In contrast, TER divides by the average reference length. This difference can sometimes result in WER being lower than the corresponding TER. Also, as can be seen in the Arabic to English results, TERp scores sometimes differ dramatically from TER scores due to normalization and tokenization differences (e.g., TERp removes punctuation prior to scoring, while TER does not).

Train\Eval	BLEU:1	BLEU:2	BLEU:3	BLEU:4	BLEU:5	NIST	TER	TERp	WER	TERpA	METR	METR-r	METR $\alpha = 0.5$	METR-r $\alpha = 0.5$
BLEU:1	77.36	56.56	41.04	29.83	21.84	9.10	61.20	61.73	68.78	57.47	42.01	59.34	41.98	59.75
BLEU:2	**77.40**	57.54	42.54	31.50	23.45	9.24	60.98	61.55	68.35	57.04	43.32	60.70	43.18	60.77
BLEU:3	77.32	**57.57**	**42.72**	31.70	23.64	9.24	60.78	61.33	68.23	56.88	42.59	59.97	42.59	60.19
BLEU:4	76.77	56.96	42.05	31.05	23.02	9.18	61.03	61.67	68.25	56.97	42.75	59.66	42.50	60.07
BLEU:5	76.98	57.27	42.63	**31.80**	**23.81**	9.37	59.45	59.94	66.91	55.42	42.69	60.09	43.08	60.67
TER	72.15	52.62	38.45	28.09	20.72	7.96	**56.05**	**56.40**	64.48	51.65	38.53	55.07	41.49	58.45
TERp	72.40	52.64	38.34	27.94	20.54	8.01	56.23	56.61	64.55	51.76	38.77	55.64	41.56	58.55
WER	74.16	54.41	39.91	29.31	21.67	8.49	56.30	56.65	**64.33**	52.02	39.41	56.53	41.54	58.61
TERpA	71.00	51.28	37.21	27.03	19.75	7.81	56.38	56.81	64.90	**51.60**	38.33	55.06	40.89	58.02
METR	73.00	53.54	39.19	28.70	21.09	8.61	68.29	69.00	73.70	64.04	43.53	60.88	41.62	59.31
METR-r	74.23	54.38	39.88	29.25	21.62	8.80	66.11	66.90	72.19	62.17	**44.00**	61.29	42.53	59.75
METR:0.5	76.81	56.83	42.09	31.21	23.26	9.37	59.42	59.98	66.91	55.23	42.77	59.90	43.02	60.68
METR-r:0.5	77.33	57.50	42.52	31.49	23.43	9.29	60.29	60.84	67.81	56.37	43.38	61.08	**43.36**	**61.27**
Combined Models														
BLEU-TER	76.32	56.70	42.17	31.45	23.59	9.17	57.10	57.57	64.95	53.02	41.09	58.31	42.82	60.28
BLEU-2TERp	75.51	55.74	41.06	30.29	22.44	8.94	56.94	57.36	65.15	52.68	40.56	57.33	42.26	59.54
BLEU+2METR	75.73	56.27	41.61	30.69	22.74	9.01	63.68	64.30	70.42	59.56	43.96	**61.44**	43.03	60.44

Table 4.1: **Chinese To English Evaluation Metric Tuning Matrix:** Chinese to English tuning set performance on MT02. In each column, cells shaded blue are better than average and those shaded red are below average. The intensity of the shading indicates the degree of deviation from average. For BLEU, NIST, and METEOR, higher is better. For edit distance metrics like TER and WER, lower is better.

Train\Eval	BLEU:1	BLEU:2	BLEU:3	BLEU:4	BLEU:5	NIST	TER	TERp	WER	TERpA	METR	METR-r	METR $\alpha = 0.5$	METR-r $\alpha = 0.5$
BLEU:1	75.98	55.39	40.41	29.64	21.60	11.94	78.07	78.71	68.28	73.63	41.98	59.63	42.46	60.02
BLEU:2	76.58	57.24	42.84	32.21	24.09	12.20	77.09	77.63	67.16	72.54	43.20	60.91	43.59	61.56
BLEU:3	**76.74**	**57.46**	**43.13**	**32.52**	**24.44**	12.22	76.53	77.07	66.81	72.01	42.94	60.57	43.40	60.88
BLEU:4	76.24	56.86	42.43	31.80	23.77	12.14	76.75	77.25	66.78	72.01	43.29	60.94	43.10	61.27
BLEU:5	76.39	57.14	42.93	32.38	24.33	12.40	75.42	75.77	65.86	70.29	43.02	61.22	43.57	61.43
NIST	76.41	56.86	42.34	31.67	23.57	12.38	75.20	75.72	65.78	70.11	43.11	61.04	43.78	**61.84**
TER	73.23	53.39	39.09	28.81	21.18	**12.73**	**71.33**	**71.70**	63.92	**66.58**	38.65	55.49	41.76	59.07
TERp	72.78	52.90	38.57	28.32	20.76	12.68	71.76	72.16	64.26	66.96	38.51	56.13	41.48	58.73
TERpA	71.79	51.58	37.36	27.23	19.80	12.54	72.26	72.56	64.58	67.30	37.86	55.10	41.16	58.04
WER	74.49	54.59	40.30	29.88	22.14	12.64	71.85	72.34	**63.82**	67.11	39.76	57.29	42.37	59.97
METR	73.33	54.35	40.28	30.04	22.39	11.53	84.74	85.30	71.49	79.47	**44.68**	62.14	42.99	60.73
METR-r	74.20	54.99	40.91	30.66	22.98	11.74	82.69	83.23	70.49	77.77	44.64	**62.25**	43.44	61.32
METR:0.5	76.36	56.75	42.48	31.98	24.00	12.44	74.94	75.32	66.09	70.14	42.75	60.98	**43.86**	61.38
METR-r:0.5	76.49	56.93	42.36	31.70	23.68	12.21	77.04	77.58	67.12	72.23	43.26	61.03	43.63	61.67
Combined Models														
BLEU:4-TER	75.32	55.98	41.87	31.42	23.50	12.62	72.97	73.38	64.46	67.95	41.50	59.11	43.50	60.82
BLEU:4-2TERp	75.22	55.76	41.57	31.11	23.25	12.64	72.48	72.89	64.17	67.43	41.12	58.82	42.73	60.86
BLEU:4+2MTR	75.77	56.45	42.04	31.47	23.48	11.98	79.96	80.65	68.85	74.84	44.06	61.78	43.70	61.48

Table 4.2: **Chinese To English Evaluation Metric Test Matrix:** Chinese to English test set performance on MT03 using models trained using MERT on MT02. In each column, cells shaded blue are better than average and those shaded red are below average. The intensity of the shading indicates the degree of deviation from average. For BLEU, NIST, and METEOR, higher is better. For edit distance metrics like TER and WER, lower is better.

Train\Eval	BLEU:1	BLEU:2	BLEU:3	BLEU:4	BLEU:5	NIST	TER	TERp	WER	METR	METR-r	METR $\alpha = 0.5$	METR-r $\alpha = 0.5$
BLEU:1	**80.22**	65.79	54.66	45.69	38.31	10.78	47.50	47.50	49.92	47.89	64.69	47.75	64.70
BLEU:2	79.97	65.94	54.98	46.09	38.73	10.80	47.21	47.21	49.37	47.71	64.90	47.96	64.91
BLEU:3	79.97	**66.00**	**55.07**	46.19	38.82	10.80	47.10	47.10	49.15	47.71	64.37	47.89	64.39
BLEU:4	79.89	65.95	55.05	46.19	38.85	10.86	46.83	46.83	48.99	48.07	64.83	47.77	65.05
BLEU:5	79.86	65.93	55.05	46.21	38.89	10.80	47.16	47.16	49.26	48.27	64.51	48.15	64.95
BLEU:6	79.68	65.83	55.03	**46.23**	**38.96**	10.92	46.34	46.34	48.63	47.72	63.95	48.11	64.46
NIST	79.49	65.70	54.86	46.01	38.70	**10.95**	46.13	46.13	48.69	47.70	63.99	48.25	65.17
TER	78.32	64.46	53.69	44.89	37.63	10.72	**45.16**	**45.16**	**47.88**	46.47	63.26	47.91	64.29
TERp	77.87	64.16	53.55	44.92	37.80	10.70	45.40	45.40	47.96	46.40	62.81	48.01	64.78
WER	78.08	64.44	53.82	45.17	38.04	10.74	45.33	45.33	47.89	46.28	62.89	48.10	64.88
TERpA	79.61	65.73	54.90	46.07	38.76	10.88	46.50	46.50	48.69	47.35	64.53	48.09	64.99
METR	75.32	61.03	50.19	41.65	34.69	9.96	53.89	53.89	55.44	**49.22**	65.77	47.20	63.57
METR-r	77.27	62.84	51.79	43.03	35.92	10.26	51.09	51.09	53.07	49.15	**65.81**	47.54	64.48
METR:0.5	79.23	65.09	54.12	45.27	38.00	10.77	47.27	47.27	49.62	49.16	65.53	**49.00**	**66.03**
METR-r:0.5	79.42	64.89	53.77	44.91	37.62	10.66	48.19	48.19	50.52	48.56	65.02	48.14	65.46
Combined Models													
BLEU-TER	79.36	65.58	54.81	46.01	38.72	10.91	45.89	45.89	48.34	47.25	64.03	48.00	64.86
BLEU-2TERpA	78.60	64.89	54.19	45.47	38.25	10.83	45.51	45.51	48.16	46.93	63.43	48.09	64.87
BLEU+2METR	79.23	64.80	53.72	44.89	37.64	10.63	48.28	48.28	50.44	48.96	65.19	48.10	65.40

Table 4.3: **Arabic To English Evaluation Metric Tuning Matrix:** Arabic to English tuning set performance on MT06. In each column, cells shaded blue are better than average and those shaded red are below average. The intensity of the shading indicates the degree of deviation from average. For BLEU, NIST, and METEOR, higher is better. For edit distance metrics like TER and WER, lower is better.

Train\Eval	BLEU:1	BLEU:2	BLEU:3	BLEU:4	BLEU:5	NIST	TER	TERp	WER	METR	METR-r	METR $\alpha = 0.5$	METR-r $\alpha = 0.5$
BLEU:1	79.90	65.35	54.08	45.14	37.81	10.68	46.19	61.04	49.98	49.74	67.79	49.19	68.12
BLEU:2	80.03	65.84	54.70	45.80	38.47	10.75	45.74	60.63	49.24	50.02	68.00	49.71	68.27
BLEU:3	79.87	65.71	54.59	45.67	38.34	10.72	45.86	60.80	49.18	49.87	68.32	49.61	67.67
BLEU:4	80.39	66.14	54.99	46.05	38.70	10.82	45.25	59.83	48.69	49.65	68.13	49.66	67.92
BLEU:5	79.97	65.77	54.64	45.76	38.44	10.75	45.66	60.55	49.11	49.89	68.33	49.64	68.19
NIST	80.41	66.27	55.22	46.32	38.98	**10.96**	44.11	57.92	47.74	48.88	67.85	**49.88**	**68.52**
TER	79.69	65.52	54.44	45.55	38.23	10.75	43.36	56.12	**47.11**	47.90	66.49	49.55	68.12
TERp	79.27	65.11	54.13	45.35	38.12	10.75	**43.36**	**55.92**	47.14	47.83	66.34	49.43	67.94
WER	79.42	65.28	54.30	45.51	38.27	10.78	43.44	56.13	47.13	47.82	66.33	49.38	67.88
METR	75.52	60.94	49.84	41.17	34.12	9.93	52.81	70.08	55.72	50.92	68.55	48.47	66.89
METR-r	77.42	62.91	51.67	42.81	35.61	10.24	49.87	66.26	53.17	**50.95**	**69.29**	49.29	67.89
METR:0.5	79.69	65.14	53.94	45.03	37.72	10.72	45.80	60.44	49.34	49.78	68.31	49.23	67.72
METR-r:0.5	79.76	65.12	53.82	44.88	37.57	10.67	46.53	61.55	50.17	49.66	68.57	49.58	68.25
Combined Models													
BLEU:4-TER	80.37	66.31	55.27	46.36	39.00	10.96	43.94	57.46	47.46	49.00	67.10	49.85	68.41
BLEU:4-2TERp	79.65	65.53	54.54	45.75	38.48	10.80	43.42	56.16	47.15	47.90	65.93	49.09	67.90
BLEU:4+2METR	79.43	64.97	53.75	44.87	37.58	10.63	46.74	62.03	50.35	50.42	68.92	49.70	68.37

Table 4.4: **Arabic To English Evaluation Metric Test Matrix:** Arabic to English test set performance on dev07 using models trained using MERT on MT06. In each column, cells shaded blue are better than average and those shaded red are below average. The intensity of the shading indicates the degree of deviation from average. For BLEU, NIST, and METEOR, higher is better. For edit distance metrics like TER and WER, lower is better.

metrics, while performing poorly on all the other metrics except NIST. While less pronounced, the same trend is also seen in the Arabic data. Interestingly enough, while the TER, TERp and standard METEOR metrics achieve good correlations with human judgments, models trained to them are particularly mismatched in the results. The edit distance models do terribly on METEOR and METEOR ranking, while METEOR and METEOR ranking models do poorly on TER, TERp, and TERpA.

4.4.1 Other Results

On the training data, I see a similar block structure within the results, but there is a different pattern among the top performers. In table 4.1, I observe that, for Chinese, the BLEU:5 model performs best on the training data according to all higher order BLEU metrics (4-7). As seen in table 4.3, on Arabic, the BLEU:6 model does best on the same higher order BLEU metrics (4-7). By rewarding higher order *n*-gram matches, these objectives actually find minima that result in more 4-gram matches than the models optimized directly to BLEU:4. However, the fact that this performance advantage disappears on the evaluation data suggests these higher order models also promote overfitting.

Models trained on additive metric blends tend to smooth out performance differences between the classes of metrics they contain. As expected, weighting the metrics used in the additive blends results in models that perform slightly better on the type of metric with the highest weight.

Training Metric	Iterations	Wall Time	Training Metric	Iterations	Wall Time
BLEU:1	13	21:57	NIST	15	78:15
BLEU:2	15	32:40	TER	7	21:00
BLEU:3	19	45:08	TERp	9	19:19
BLEU:4	10	24:13	TERpA	8	393:16
BLEU:5	16	46:12	WER	13	33:53
BL:4-TR	9	21:07	BL:4-2TRp	8	22:03
METR	12	39:16	METR 0.5	18	42:04
METR R	12	47:19	METR R:0.5	13	25:44

Table 4.5: **Training Time:** Chinese to English MERT iterations and training times, given in hours:mins and excluding decoder time, when tuning to different evaluation metrics.

Table 4.5 reports training times for select Chinese to English models. Training to TERpA is very

computationally expensive due to the implementation of the paraphrase-table. The TER family of metrics tends to converge in fewer MERT iterations than those trained on other metrics such as BLEU, METEOR or even WER. This suggests that the learning objective provided by these metrics is either easier to optimize or they more easily trap the search in local minima.

4.4.2 Model Variance

One potential problem with interpreting the results above is that learning with MERT is generally assumed to be noisy, with different runs of the algorithm possibly producing very different models. I explore to what extent the results just presented were affected by noise in the training procedure. I perform multiple training runs using select evaluation metrics and examining how consistent the resulting models are. This also allows us to determine whether the metric used as a training criterion influences the stability of learning. For these experiments, Chinese to English models are trained 5 times using a different series of random starting points. As before, 20 random restarts were used during each MERT iteration.

Train\σ	BLEU:1	BLEU:3	BLEU:4	BLEU:5	TERp	WER	METR	METR α:0.5
BLEU:1	**0.17**	0.56	0.59	0.59	0.36	0.58	0.42	0.24
BLEU:3	0.38	0.41	0.38	0.32	0.70	0.49	0.44	0.33
BLEU:4	0.27	0.29	0.29	0.27	0.67	0.50	0.41	0.29
BLEU:5	**0.17**	0.14	0.19	0.21	0.67	0.75	0.34	0.17
TERp	1.38	2.66	2.53	2.20	1.31	1.39	1.95	1.82
WER	0.62	1.37	1.37	1.25	1.31	1.21	1.10	1.01
METR	0.80	0.56	0.48	0.44	3.71	2.69	0.69	1.10
METR:0.5	0.32	**0.11**	**0.09**	**0.11**	**0.23**	**0.12**	**0.07**	**0.11**

Table 4.6: **Model Variation:** MERT model variation for Chinese to English when training to different evaluation metrics. I train five models to each metric listed above. The collection of models trained to a given metric is then evaluated using each of the training metrics. I report the resulting standard deviations for the collections. The collection with the lowest variance is bolded.

In table 4.6, models trained to BLEU and METEOR are relatively stable, with the METEOR:0.5 trained models being the most stable. The edit distance models, WER and TERp, vary more across training runs, but still do not exceed the interesting cross metric differences seen in table 4.2. However, the instability of WER and TERp, with TERp models having a standard deviation of 1.3 in TERp and 2.5 in BLEU:4, make

Chinese			Arabic		
Model Pair	% Pref	*p*-value	Model Pair	% Pref	*p*-value
METR R vs. TERp	60.0	0.0028	**BLEU:4** vs. METR R	62.0	< 0.001
BLEU:4 vs. TERp	57.5	0.02	NIST vs. TERp	56.0	0.052
NIST vs. TERp	55.0	0.089	BLEU:4 vs. METR:0.5	55.5	0.069
NIST vs. TERpA	55.0	0.089	BLEU:4 vs. METR	54.5	0.11
BLEU:4 vs. TERpA	54.5	0.11	METR R:0.5 vs METR R	54.0	0.14
BLEU:4 vs. METR R	54.5	0.11	NIST vs. BLEU:4	51.5	0.36
METR:0.5 vs. METR	54.5	0.11	WER vs. TERp	51.5	0.36
METR:0.5 vs. METR R	53.0	0.22	METR:0.5 vs METR	51.5	0.36
METR vs. BLEU:4	52.5	0.26	TERp vs. BLEU:4	51.0	0.42
BLEU:4 vs. METR:0.5	52.5	0.26	BLEU:4 vs. METR R:0.5	50.5	0.47
METR vs. TERp	52.0	0.31			
NIST vs. BLEU:4	52.0	0.31			
BLEU:4 vs. METR R:0.5	51.5	0.36			
WER vs. TERp	51.5	0.36			
TERpA vs. TERp	50.5	0.47			

Table 4.7: **Human Preferences:** Select pairwise preference for models trained to different evaluation metrics. For A vs. B, *preferred* indicates how often A was preferred to B. I bold the better training metric for statistically significant differences.

them risky metrics to use for training.

4.5 Human Evaluation

The best evaluation metric to use during training is the one that ultimately leads to the best translations according to human judges. I perform a human evaluation of select models using Amazon Mechanical Turk, an online service for cheaply performing simple tasks that require human intelligence. To use the service, tasks are broken down into individual units of work known as human intelligence tasks (HITs). HITs are assigned a small amount of money that is paid out to the workers that complete them. For many natural language annotation tasks, including machine translation evaluation, it is possible to obtain annotations that are as good as those produced by experts by having multiple workers complete each HIT and then combining their answers (Snow et al., 2008; Callison-Burch et al., 2006).

I perform a pairwise comparison of the translations produced for the first 200 sentences of the Chinese to English test data (MT03) and the Arabic to English test data (dev07). The HITs consist of a pair of machine translated sentences and a single human generated reference translation. The reference is chosen at random

from those available for each sentence. Capitalization of the translated sentences is restored using an HMM based truecaser (Lita et al., 2003). Turkers are instructed to "⋯select the machine translation generated sentence that is easiest to read and best conveys what is stated in the reference". Differences between the two machine translations are emphasized by being underlined and bold faced.[8] The resulting HITs are made available only to workers in the United States, as pilot experiments indicated this results in more consistent preference judgments. Three preference judgments are obtained for each pair of translations and are combined using weighted majority vote.

As shown in table 4.7, in many cases the quality of the translations produced by models trained to different metrics is remarkably similar. Training to the simpler edit distance metric WER produces translations that are as good as those from models tuned to the similar but more advanced TERp metric that allows for swaps. Similarly, training to TERpA, which makes use of both a paraphrase-table and edit costs tuned to human judgments, is no better than TERp.

For the Chinese to English results, there is a statistically significant human preference for translations that are produced by training to BLEU:4 and a marginally significant preference for training to NIST over the default configuration of TERp. This contrasts sharply with earlier work showing that TER and TERp correlate better with human judgments than BLEU (Snover et al., 2009; Przybocki et al., 2008; Snover et al., 2006). While it is assumed that, by using MERT, "improved evaluation measures lead directly to improved machine translation quality" (Och, 2003), these results show improved correlations with human judgments are **not always** sufficient to establish that tuning to a metric will result in higher quality translations. In the Arabic results, I see a similar pattern where NIST is preferred to TERp, again with marginal significance. Strangely, however, there is no real difference between TERp vs. BLEU:4.

For Arabic, training to ranking METEOR is worse than BLEU:4, with the differences being very significant. The Arabic results also trend toward suggesting that BLEU:4 is better than either standard METEOR or METEOR α 0.5. However, for the Chinese models, training to standard METEOR and METEOR α 0.5 is about as good as training to BLEU:4. In both the Chinese and Arabic results, the METEOR α 0.5

[8] I emphasize relative differences between the two translations rather than the difference between each translation and the reference in order to avoid biasing evaluations toward edit distance metrics.

models are at least as good as those trained to standard METEOR and METEOR ranking. In contrast to the cross evaluation metric results, where the differences between the α 0.5 models and the standard METEOR models were always fairly dramatic, the human preferences suggest there is often not much of a difference in the true quality of the translations produced by these models.

4.6 Conclusion

Training to different evaluation metrics follows the expected pattern whereby models perform best on the same type of metric used to train them. However, models trained using the *n*-gram based metrics, BLEU and NIST, are more robust to being evaluated using the other metrics. Edit distance models tend to do poorly when evaluated on other metrics, as do models trained using METEOR. However, training models to METEOR can be made more robust by setting α to 0.5, which balances the importance the metric assigns to precision and recall.

The fact that the WER, TER and TERp models perform very similarly suggests that current phrase-based translation systems lack either the features or the model structure to take advantage of swap edit operations. The situation might be improved by using a model that does a better job of both capturing the structure of the source and target sentences and their allowable reorderings, such as a syntactic tree-to-string system that uses contextually rich rewrite rules (Galley et al., 2006), or by making use of larger more fine grained feature sets (Chiang et al., 2009) that allow for better discrimination between hypotheses.

Human results indicate that edit distance trained models such as WER and TERp tend to produce lower quality translations than BLEU or NIST trained models. Tuning to METEOR works reasonably well for Chinese, but is not a good choice for Arabic. The newer RYPT metric (Zaidan & Callison-Burch, 2009), which directly makes use of human adequacy judgments of substrings, could obtain better human results than the automated metrics presented here. However, the metric is less attractive for running experiments on different system variations due to the increased cost of tuning. If the metric is used for tuning, performance gains still will likely be sensitive to how the mechanics of the metric interact with the structure and feature set of the decoding model being used.

BLEU and NIST's strong showing in both the machine and human evaluation results indicates that

they are still the best general choice for training model parameters. This emphasizes that improved metric correlations with human judgments do not imply that models trained to a metric will result in higher quality translations. Future work on developing new evaluation metrics should then explicitly explore the translation quality of models trained to them.

One of the reasons why training to different evaluation metrics like BLEU, METEOR and TER did not have much of an impact on human preferences for the translations produced by the resulting models may have been that on some level all of these evaluation metrics are relatively similar. That is, they all essentially score a translation based upon substring overlap, with the more advanced metrics allowing for fuzzy matching based on lexical semantics and paraphrasing. The next chapter will present results on when a system is tuned to a more advanced evaluation metric that operates by performing a deeper semantic analysis of the system's translations and the references.

Chapter 5

Improved Translation Quality Using Textual Entailment

5.1 Introduction

As seen in the last chapter, tuning to different lexical level evaluation metrics does not seem to make a significant difference in the preferences of human judges for the resulting translation systems. One reason this might be the case is that what is being measured by all of these metrics is essentially the same. These metrics all score translations based on the number of overlapping strings that are present in both the system translations and in the reference translations. Newer metrics do a better job of matching strings that mean the same thing but that are worded differently. However, the amount of variation they allow is limited to matching words with similar meanings or very local paraphrasing.

This chapter examines tuning to a more advanced evaluation metric that actually performs a deep semantic analysis of the two translations being compared to each other. It does this by running the translations through a textual entailment detection system. Given two sentences, an entailment system attempts to accurately determine if one sentence can be plausibly "inferred" from the other. Relevant to translation evaluation, if two sentences both entail each other, then it is reasonable to conclude a fluent speaker would consider that the two sentences both convey the same meaning, and thus could be equally good translations of some original source material. In order to accurately detect such entailment relationships, the entailment system makes use of a wide range of rich linguistic representations that provide evidence for lexical, syntactic, and structural semantic matches and mismatches.

However, using such an evaluation metric for training presents a number of new challenges. First, the metric is significantly slower than other simpler metrics such as BLEU, TER, or METEOR. Using the metric

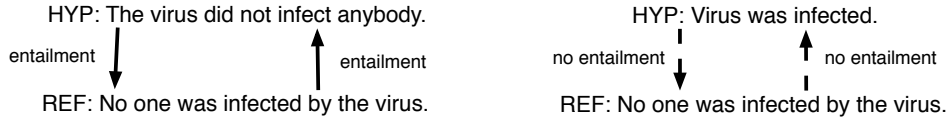

Figure 5.1: **Mutual Entailment of Translations:** Entailments between a machine translation system hypothesis and a reference translation for good translations (right) and bad translations (left). Figure courtesy of (Padó et al., 2009a).

for training requires careful adaptation of the training produce. Second, much of what is being measured by the new metric is not readily captured by existing machine translation decoding models. Such a mismatch reduces the model's capacity to fit and benefit from the metric.

The remainder of this chapter is organized as follows. Section 5.2 presents the entailment system and its adaptation into a machine translation evaluation metric. Section 5.3 details the integration of the entailment based metric into MERT. Section 5.4 describes my experiments, and sections 5.5 and 5.6 present and discuss the key result: The entailment-based model produces translations that are perceived by human raters as significantly better than those constructed by models trained using TER and BLEU.

5.2 Textual Entailment and Machine Translation Evaluation

As introduced by Dagan et al. (2005), the textual entailment recognition task attempts to capture the inferences people would make when reading a passage. The entailments do not need to strictly logically follow from a passage, but rather also include anything that is likely to be true given the passage. For example, the passage "John has $20" does not logically imply that "John has less than $30". John could have $40 dollars making the passage correct, as someone with $40 also has $20. But with $40 in John's pocket, the statement "John has less than $30" would be incorrect. However, after reading the passage most reasonable speakers would in fact make the inference that "John has less than $30", as conversational maxims dictate that if John actually has more than $20 a non-deceptive speaker or writer should have actually made this information explicit (Levinson, 1983).

To perform the entailment task, a system is asked to label a series of passage and hypothesis pairs as being valid entailments or not. This is similar to the work done by a machine translation evaluation

metric. However, a translation evaluation metric should not only ensure that everything stated in a system's hypothesis is supported by the reference, but also that all of the information in the reference is captured by the hypothesis. As shown in figure 5.1, this suggests that the entailment task can be mapped onto machine translation evaluation by simply checking for bi-directional entailment.

Early entailment systems attempted to use bag-of-words overlap or other surface-based methods (Monz & de Rijke, 2001; Glickman & Dagan, 2005). These methods represent a strong baseline, but have lately been consistently outperformed by systems built on a richer feature set that encodes lexical, structural, and more sophisticated semantic properties of the sentence pairs (Bar-Haim et al., 2007; Marsi et al., 2007; Roth & Sammons, 2007; Zanzotto et al., 2007). In this respect, there is a strong parallelism to the situation in translation evaluation, with metrics like BLEU, NIST, METEOR and TER that base their judgment on overlapping word sequences taking the place of robust, but ultimately simplistic bag-of-words models. While these methods have the advantage of being relatively fast, their superficial analysis risks erroneously assigning a low score to translations that are good but liberally reworded paraphrases of a reference translation.

In contrast, the deeper analysis performed by entailment-based metrics is computationally expensive, but also hopefully results in a more accurate assessment of semantic equivalence. The deeper analysis performed by state-of-the-art entailment systems allows the resulting metric to capture aspects of translation quality that are missed by surface-level metrics, such as matches or mismatches in argument structure, the quality of paraphrases with little surface overlap, or the preservation of long-distance dependencies.

These intuitions have been confirmed by recent work involving the use of an entailment recognition system for machine translation evaluation (Padó et al., 2009a; Padó et al., 2009b). It was found that the entailment based metric tended to outperform a set of popular translation evaluation metrics (BLEU, NIST, TER, and METEOR) in predicting human assessments of translation quality across a wide range of corpora. Moreover, it was found that the entailment based metric was best at discriminating between translations on the higher end of the scale, that is, between good translations and even better translations. The latter is critical, since further progress in machine translation will require evaluation metrics that capture fine-grained distinctions in translation quality. This makes entailment based metrics a promising possible replacement for surface level metrics like BLEU, TER, and METEOR for use in model tuning with MERT.

5.2.1 Stanford Entailment Recognizer Metric

This work uses the entailment based evaluation metric (TEval) described by Padó et al. (2009a). The metric operates by running the Stanford Entailment Recognizer (SER, MacCartney et al. (2006)) in both directions over a reference and a candidate translation. The features extracted during both runs of the SER system are then combined in a linear regression model trained to predict human assessments of translation quality. The resulting TEval scoring model produces sentence level scores that range from 0 to 7, with 7 corresponding to a perfect translation and 0 representing a completely unacceptable one.

As illustrated in figure 5.2, the SER system uses a pipelined approach. It begins with a deep linguistic analysis on incoming sentence pairs. The Stanford PCFG parser is run over the translations, and the resulting phrase-structure parse trees are converted into typed dependency graphs using a series of rule-based transformations (de Marneffe et al., 2006). Subsequently, an alignment is created between the nodes in the two graphs, which takes into account both the structure of the graphs and the semantic similarity of the nodes being aligned.

From the aligned graphs, the system extracts features that suggest either the presence or absence of an entailment relationship. One of the most basic, but yet useful, features simply reproduces the graph alignment score assigned to the pair, as it should generally be easier to align hypotheses that follow from a passage than it is to align those that do not follow. Most of the other features examine the aligned graphs using simple linguistically motivated rules that suggest the presence or absence of an entailment relationship. Padó et al. (2009a) groups such features into four classes. The *semantic compatibility* features check whether or not meaning is preserved by the aligned nodes. The *insertion and deletion* features attempt to capture what sorts of material can be inserted or deleted from the two sentences without breaking the entailment relationship. *Preservation of reference* features attempt to verify that both sentences seem to be referring to the same events and participants. *Structural alignment features* attempt to verify the compatibility of grammatically important parts of each sentence such as the main verb and its argument structure. To increase robustness in case the linguistic analysis fails, the TEval metric also includes as features a set of *traditional machine translation evaluation metrics* based on variants of the BLEU, NIST, TER, and METEOR scores.

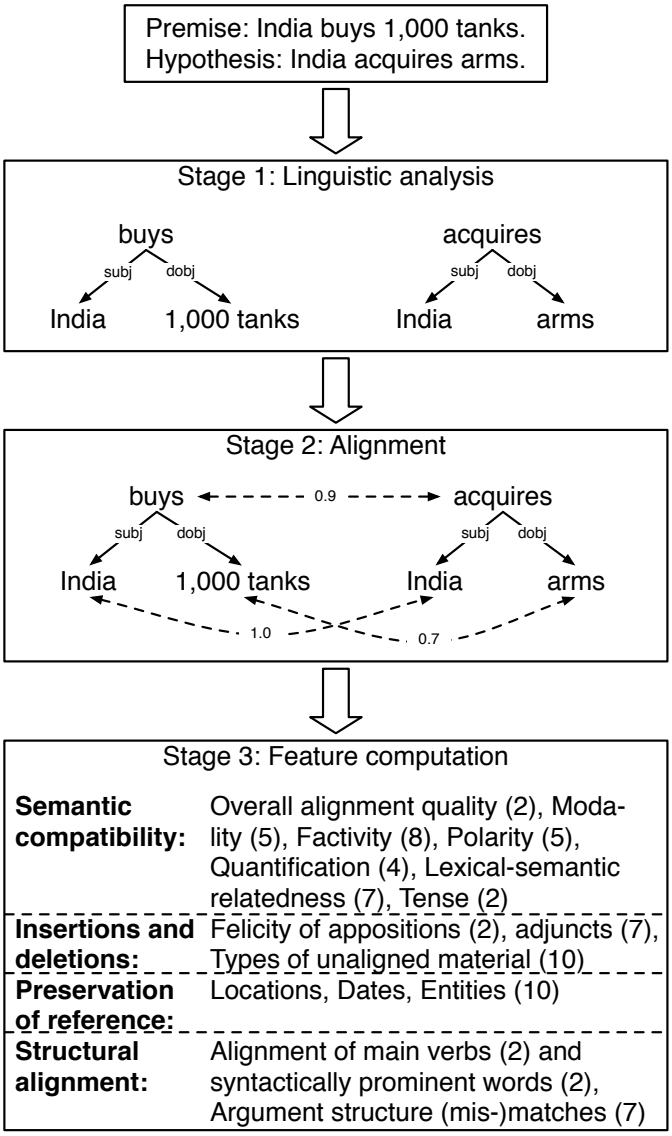

Figure 5.2: **Stanford Entailment Recognizer:** The pipelined approach used by the Stanford entailment recognizer to analyze sentence pairs and determine whether or not an entailment relationship is present. The entailment recognizer first obtains dependency parses for both the passage and the hypothesis. These parses are then aligned based upon lexical and structural similarity between the two dependency graphs. From the aligned graphs, features are extracted that suggest the presence or absence of an entailment relationship. Figure courtesy of (Padó et al., 2009a).

5.3 Using Textual Entailment within MERT

One drawback of the TEval metric is the amount of computation required for each sentence pair. As it was originally developed, it required approximately 6 seconds per sentence pair being evaluated. However, a

single iteration of MERT training can require millions of calls to the selected evaluation metric. For instance, I found that, when training on MT02 (878 sentences) using 100-best lists, the final iteration of MERT required around 4.6 million invocations of the evaluation metric. Assuming there are 4 reference translations, a naïve embedding of the TEval metric that would perform a sentence pair analysis with each of the 4 references for each call to the evaluation metric by MERT would result in this round of training alone requiring almost 3.6 years to complete.

A small number of strategic changes were performed in order to make learning with TEval tractable. First, since the most costly analysis performed by TEval is the initial parse of the two sentences, I introduced caching of the parse trees. Critically, this not only prevents each pair of sentences from being parsed twice (i.e., once for each entailment direction), but it also prevents reference translations from being repeatedly parsed for each entry on the *n*-best lists that they are being compared with. Second, when presented with multiple reference translations, I only calculate the TEval score using the reference for which the candidate translation has the best TER score. Third, I introduced caching of the scores calculated by TEval for each unique sentence pair. Thus, during each iteration of MERT TEval was only run over the new unique *n*-best list entries that were not seen during previous iterations. This resulted in significant savings since even during early iterations of training typically 60 to 90% of the *n*-best entries were either duplicates or were seen during prior iterations.[1]

Even with these changes, TEval still proved to be somewhat slow. On an AMD Opteron 2216 and training on 878 sentences using 100-best lists, even early iterations of MERT using TEval took on average 4 days to complete.[2] Since 10 iterations or more are often performed prior to convergence, MERT with TEval would still take well over a month to complete. One approach would be to simply apply more computing hardware to the problem, since the sentence pairs evaluated by the TEval metric could be trivially evaluated in parallel on multiple machines. With as little as two dozen dedicated CPUs, the amount of time required for each iteration would be brought down to a manageable few hours.

[1] Such caching of evaluation scores for sentence pairs also led to a 5× speed up in the time required to run each iteration of MERT when using TER as the evaluation metric.

[2] Results for training MERT on the complete 100-best lists are not presented here, as the system took prohibitively long to finish training.

However, such an approach could only be used at somewhat resource rich sites. A more accessible alternative, and the one that is explored here, is to break training down into two manageable steps. The first is to train an initial model using one of the traditional translation evaluation metrics. In the second step, the resulting model weights are used as the starting point for MERT using the TEval metric and small n-best lists (i.e., with $n = 10$). While such small n-best lists would not be sufficient to train a new model from scratch, I found that they do allow for successful adaptation of the existing model to the new evaluation metric while minimizing the computation cost. With these changes, each MERT iteration required approximately 11 hours.

5.4 Experiments

Experiments were performed using the Chinese-English evaluation data provided for NIST MT eval 2002, 2003, and 2005. MT02 was used as a dev set for MERT learning, while MT03 and MT05 were used as the test sets. I contrasted the performance of a model trained using TEval with those trained using the popular BLEU and TER evaluation metrics. I also explored the effect of the initial starting point used for the TEval metric by training models using initial weights based on both the BLEU and TER trained models.

5.4.1 System

Experiments were run using the Phrasal machine translation package (Cer et al., 2010b). During decoding I made use of a stack size of 100, set the distortion limit to 6, and retrieved 20 translation options for each unique source phrase.

The phrase-table was built using 1,140,693 sentence pairs sampled from the GALE Y2 training data. The Chinese data was word segmented using the GALE Y2 retest release of the Stanford CRF segmenter (Tseng et al., 2005). Phrases were extracted using the typical approach described in (Koehn et al., 2003) of running GIZA++ (Och & Ney, 2003) in both directions and then merging the alignments using the grow-diag-final heuristic. From the merged alignments, a bidirectional lexical reordering model conditioned on the source and the target phrases (Tillmann, 2004; Koehn et al., 2007) was also extracted. A 5-gram language model was created using the SRI language modeling toolkit (Stolcke, 2002) and trained using the Gigaword

System	MT02 (dev)			MT03 (test)			MT05 (test)		
	BLEU	TER	TEvl	BLEU	TER	TEvl	BLEU	TER	TEvl
BLEU	**31.5**	58.6	4.54	**32.0**	57.4	4.58	**30.3**	58.4	4.49
TER	30.4	**56.8**	4.59	31.1	**56.1**	4.62	29.0	**56.6**	4.55
TEvl-BLEU	30.7	57.3	4.62	31.2	56.9	4.67	29.8	57.3	4.60
TEvl-TER	30.1	57.1	**4.66**	30.8	56.4	**4.70**	29.5	57.0	**4.63**

Table 5.1: **Tuning to TEval:** Performance of models trained using MERT with BLEU, TER, TEval using the BLEU model as a starting point, and TEval using TER as a starting point. The bolded score indicates the best performing system according to each metric. For BLEU and TEval, higher scores are better, while for TER lower scores are preferable.

corpus and English sentences from the parallel data.

5.5 Results

As seen in table 5.1, both models trained using TEval outperform those trained using either BLEU or TER on both the dev set and the test sets when evaluated by TEval. It is interesting to note that the model trained using TER does better on TEval than the one trained using BLEU. This might be attributed to the fact that TER has been found to correlate more highly with human judgments than BLEU (Snover et al., 2006).

Notice that TER also provides a better starting point for tuning the model to TEval, as the TEval model that was trained using the TER model weights as a starting point systematically outperforms the TEval-BLEU model. Presumably, with larger *n*-best lists, MERT with TEval would result in models that obtain similar performance regardless of whether BLEU or TER was used as the starting point. However, using a TER-based starting point would likely still be more desirable, as it would probably lead to fast convergence.

5.6 Human Assessments using Mechanical Turk

In order to validate that the improvements obtained by training toward the TEval metric actually correspond to true improvements in translation quality, I also performed a cost-effective manual comparison of the systems using Amazon's Mechanical Turk. Recently, it has been shown that for certain NLP-related annotation tasks, expert-level labeling can be obtained by simply averaging the work performed by a small

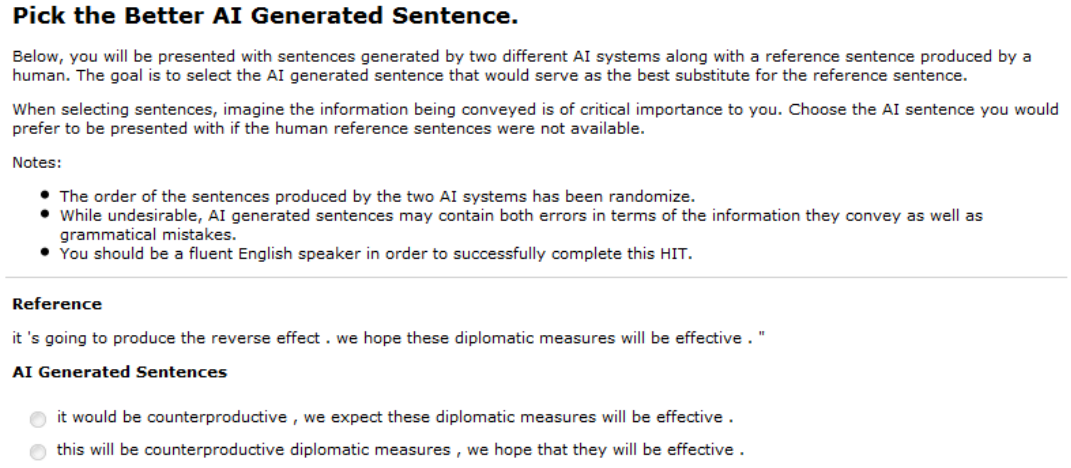

Figure 5.3: **Translation Evaluation Presentation:** Human Intelligence Task (HIT) sentence pair presented to Amazon Mechanical Turk annotators.

number of people on the Mechanical Turk service (Snow et al., 2008).

Manually evaluation of machine translations is typically done on a 5 or 7 point Likert scale (Przybocki & Bronsart, 2008). This makes it possible to assess the performance of a large number of systems without having to make a sizable number of paired comparisons between each system. However, for my purposes, I elect to formulate the manual evaluation task as a paired comparison between systems. Since the systems I will be comparing under different experimental conditions are very similar, I believe a pair presentation is desirable in order to detect subtle differences in translation quality that may only be apparent when two alternative translations are presented simultaneously.

As shown in figure 5.3, the human evaluators were presented with the pair of sentences produced by two different system configurations along with a single reference translation. People were then told to select the sentence that best matched the reference translation. Specifically, people were instructed to imagine that the information being conveyed by the sentence was of some importance to them and that they should choose the sentence they would prefer to be presented with if the human reference sentence was not available. Each sentence pair was presented to five human evaluators at the cost of USD 0.01 per pair.

The output of the best performing TEval trained system (i.e., using a TER-trained model as a starting point) was manually compared to that produced by systems trained using TER and BLEU. Human evalu-

ation required approximately 48 hours per data set, with annotators completing about 100 translation-pair comparisons per hour. Inter-annotator agreement was estimated by first creating a single meta-annotator that selected translations based on the majority vote over the actual annotations. Agreement with the meta-annotator as measured by the Pearson correlation coefficient was found to be 0.71. While not terrible, such a level of agreement does confirm the need for multiple annotations per sentence pair.

As shown in table 5.2, the gains obtained by using the TEval metric during training do in fact result in translations that tend to be preferred by human judges. While the strength of the effect is modest, the results are statistically significant. Moreover, it is expected that more substantial gains could be obtained by providing the translation model with a richer feature set that addresses more of the phenomena measured by the TEval evaluation metric.

System Pairs	% TEval Preferred	p
TEval vs. BLEU	52.9	< 0.001
TEval vs. TER	51.7	< 0.01

Table 5.2: **Human Pairwise Preferences:** Comparison of human pairwise preference for translations produced by models train to TEval vs. BLEU and TEval vs. TER and degree of statistical significance.

5.7 Analysis

To help understand why a system trained by the TEval metric produces better sentences, I examined the sentences in the MT02 dev set produced by the BLEU, TER, and TEval trained systems. I found that the sentences produced by the TEval systems that scored more highly on the new metric than their BLEU or TER trained counter parts, exhibit fewer drops or mistranslations of structurally important words and also do a better job of preserving argument structure.

One example is shown in figure 5.3, in which both the BLEU and TEval trained systems tend to drop words that are present in the reference translation. But the words they drop have very different roles in the meaning of the sentence. For example, the BLEU trained system not only fails to represent that the interior minister is conveying information about what has happened to the Italian government adviser but it also makes the erroneous claim that the interior minister was the one who was shot.

Ref	Interior minister confirms senior adviser to the Italian government was shot by the ” red brigade ”	TEval
BLEU	The senior adviser to the interior minister was shot dead ” .	3.73
TEval	Interior minister said the senior adviser was shot .	4.15

Table 5.3: **TEval vs. BLEU Tuning Error Analysis:** Development set translations produced by a human translator (Ref), a system trained to TEval and a system trained to BLEU. Notice that the TEval system does a better job of preserving the meaning of the reference translation.

While the BLEU score treats all word deletions as equivalent, the TEval metric specifically penalizes mismatches in prominent words such as the main verb. Additionally, even though “said” can be seen as largely preserving the meaning of the translation, BLEU would not reward such a production unless it was also present in one of the reference translations and for this particular example all of the reference translations contain either “confirms” or “confirmed”.

The TEval metric also searches for argument mismatches and, as seen above, penalizes sentence pairs where an action is either being performed by or performed on the wrong entities. In contrast, BLEU, TER and other lexical semantics based evaluation metrics like METEOR only measure mismatches in argument structure using essentially the relative sequence position of the arguments. Moreover, for these word-sequence based metrics, mistakes in argument structure are counted no more heavily than other less serious errors or even allowable substitutions not captured in at least one the of the reference translations.

5.8 Conclusion

Automated measures of translation quality play a significant role in machine translation research. Not only do they serve as the yard stick with which progress is measured, but they also play a critical role in the parameterization of models. If some aspect of translation quality is not being adequately measured by the metric being optimized using MERT, it is expected that the final model’s performance on that phenomenon will be compromised. This will even be the case if the model’s structure is capable of producing better quality translations, as the feature weights will only be trained to optimize performance according to what is rewarded by the selected metric.

This could lead to underestimating the value of certain features in a translation system. That is, let's assume I develop a novel feature which has the capacity to significantly improve the true quality of the translations produced by a model. However, if the translation characteristics it could improve are not adequately measured by the evaluation metric, then experimental results using the metric will likely not show sizable improvements over the scores obtained by existing systems. Worse, the new feature will be weighted during MERT by the apparent value it provides according to the flawed metric and thus is likely to be underweighted by the model, and given little influence in deciding the best translation candidate. Consequently, even human assessment of the system will underestimate the feature's value.

While training toward better more sophisticated metrics holds significant promise, their increased computational requirements makes it essential to be strategic about how one approaches using them to train models. The staged training technique presented here is one way in which a model can be efficiently tuned toward such a computationally intensive metric. I anticipate that future work will look at other alternatives. One promising avenue for such investigations may be developing strategies for selecting the most informative subset of the data to be used during each iteration of training.

Chapter 6

Model Learning without Minimum Error Rate Training

6.1 Introduction

While minimum error rate training (MERT) allows machine translation models to be fit precisely to automated evaluation measures of machine translation quality, we have seen in chapter 4 that there is not necessarily a clear correspondence between tuning to many popular evaluation metrics and human preferences for the resulting translations. This raises the question of whether or not it is worthwhile exactly fitting evaluation metrics. This chapter investigates training machine translation models using standard techniques widely applied to other natural language processing and machine learning tasks and contrasts the resulting models to those produced by MERT.

For much of the work on natural language processing tasks, models are trained to a general purpose objective function (e.g., maximum conditional likelihood for log-linear models) that have no relationship to whatever metric will be used to ultimately evaluate the resulting model. Recent work using large-margin methods such as support vector machines for interdependent and structured output spaces (ISOSVM), margin-infused relaxed algorithm (MIRA), and max-margin Markov networks (M^3) can make use of an evaluation metric to fine tune the separation in model scores between correct and incorrect states, with the idea being that you want either a greater separation or greater margin violation penalty for incorrect states with very low evaluation metric scores. Such techniques allow a model to be adapted to a specific evaluation metric, while still being trained by optimizing a well behaved large-margin objective function and without directly walking the potentially complex error surface defined by the evaluation metric.

Since these approaches seem to work reasonably well for other domains, it is rather surprising that

they are not widely used in machine translation as well. The reason for this is that, when MERT was first introduced, one of the key findings was that models trained using MERT to a variety of different evaluation metrics significantly outperformed log-linear models trained to a standard conditional likelihood objective on whatever evaluation metric was used to train the MERT models (Och, 2003). Since the models were only evaluated using automated metrics, and in light of the results presented in chapter 4, there is some question about whether or not the MERT models were actually better or if they just did a better job of fitting the idiosyncratic preferences of each evaluation metric.

Even if MERT does allow us to usefully eke out the best possible performance out of models, it also brings with it a number of limitations. Unlike maximum conditional likelihood log-linear models or large-margin methods, MERT does not scale well to large numbers of features. Using feature sets with massive numbers of binary features has come to dominate research work on many other natural language processing tasks. Using MERT thus creates a division between the sort of feature engineering that works well for other domains and the feature engineering that must be done for statistical machine translation systems.

Recently Chiang et al. (2009) and Chiang et al. (2008) demonstrated that it is possible to train machine translation systems using the MIRA algorithm and obtain performance that is better than that of models trained using MERT. Using a typical feature set for hierarchical machine translation models, Chiang et al. (2009) and Chiang et al. (2008) both show that MIRA trained models outperform those produced by MERT.[1] Moreover, Chiang et al. (2009) showed that the performance gap between the MERT and MIRA trained models can be increased further by introducing large numbers of discriminative features that cannot be effectively trained using MERT. While these findings have created significant interest in using MIRA for training machine translation models, other researchers have not yet broadly replicated Chiang et al. (2009) and Chiang et al. (2008)'s results. As such, MERT still remains the most popular method for training machine translation models.

In this chapter, I investigate the use of multiple alternatives to MERT including training using log-

[1] In Chiang et al. (2008)'s experiments, the MERT model outperforms the MIRA model on the newswire portion of the NIST MT06 test set. However, the MIRA trained model does much better on the newsgroup data, allowing it to achieve a higher score on the entire dataset. Since both models were trained on newswire, this suggests the MIRA models generalizes better than the MERT model.

linear models to a conditional log-likelihood objective, ISOSVM, M^3 networks, and MIRA. It is shown that all of these methods perform as well or better than MERT. These findings will hopefully encourage future work on machine translation to no longer fell bound to the MERT algorithm, but rather researchers will be able to freely choose from available machine learning frameworks just as they do for other natural language processing domains.

In the next section, I review all of the machine learning techniques investigated in this chapter and describe how they are used here for training machine translation models. I then present experimental results for each of the learning methods for both Chinese to English and Arabic to English translation. Finally, I provide an analysis of my results in light of Och (2003)'s original experiments that seemed to suggest that MERT radically outperforms mainstream machine learning algorithms such as log-linear models trained to a conditional log-likelihood objective.

6.2 Learning Algorithms

This section briefly reviews the learning algorithms explored in this chapter, as well as how I use the algorithms for training machine translation models. The algorithms explored here include a number of popular machine learning techniques, including a variant of support vector machines for structured prediction problems, max-margin Markov models, log-linear training, and MIRA. The biggest difference between using these techniques for machine translation and other learning tasks is that machine translation requires heuristic selection of learning targets.

6.2.1 Selecting Learning Targets

One of the advantages of MERT is that it allows training toward human generated reference translations without explicitly choosing learning targets. For other discriminative learning techniques, learning operates by maximizing the score assigned to specific positive learning targets as compared to the score assigned other labels or outputs that can be generated by system. However, machine translation systems have difficulty producing the *exact* human generated reference translations, due to either not having the appropriate phrase pairs in the system's phrase table or re-ordering limitations such as those imposed by the system's

distortion limit. Often, the system will either have no way of producing the human reference translations or will only be able to generate the reference translations using some undesirable derivation, such as translating punctuation into content words (Liang et al., 2006a). To use discriminative learning techniques that require explicit positive learning examples, translation targets are typically chosen heuristically from system generated *n*-best lists rather than training directly towards human generated targets.

However, some metrics such as the BLEU score cannot be effectively applied to individual sentences. For example, applying the BLEU:*N* score to individual sentences results in all sentences that do not have at least one *n*-gram match of length *N* receiving a score of 0. This means that the standard BLEU:4 configuration of the metric would not assign an informative evaluation metric score to any sentences that did not match at least one 4-gram in the reference translations.

Prior work has addressed this by selecting learning targets using smoothed variants of evaluation metrics that can be applied at the sentence level, such as the smooth BLEU score (Liang et al., 2006a; Watanabe et al., 2006; Chiang et al., 2008; Chiang et al., 2009). In this work, I introduce a new approach that selects learning targets by hillclimbing the corpus level evaluation metric score.

While the learning targets selected by both techniques should be similar, the corpus level hill climbing technique has the advantage that it can be used with *arbitrary* evaluation metrics without needing to devise any metric specific smoothing techniques. Corpus level hillclimbing also eliminates the risk of systematic artifacts being accidentally introduced by smoothed variants of a corpus level evaluation metric.

6.2.1.1 Hillclimbing Evaluation Metrics

It is not possible to test the evaluation score assigned to all possible combinations of learning targets that can be selected from *n*-best lists, since the number of possible combinations scales exponentially in the size of the corpus being translated.[2] To avoid testing all possible combinations, learning targets are selected by hillclimbing as follows.

First, the system loops over the *n*-best lists and does a greedy selection of sentences that maximize the corpus level score assigned by the evaluation metric. When using BLEU, this means that targets are selected

[2] Using *n*-best lists of length *n* and a corpus containing *l* sentences, there are $O(l^n)$ possible ways to choose the learning targets.

at random until the hillclimbing algorithm runs across an *n*-best list where it is possible to select a learning target that matches one of the 4-grams in the reference translations.

After selecting an initial set of potential learning targets, the system then iterates over the *n*-best lists attempting to increase the evaluation metric score by swapping the learning targets with other *n*-best list entries. Hillclimbing stops once the system makes one complete pass over the *n*-best lists without changing any of the learning targets. Convergence usually occurs after 2 or 3 passes over the *n*-best lists.

6.2.2 Margin Infused Relaxed Algorithm

The margin infused relaxed algorithm (MIRA) is an online learning technique. Rather than optimizing some global measure of the merit or fit of the model to training data, online algorithms operate by examining the system's performance on one data point at a time making incremental adjustments to the model weights. Learning in MIRA operates by making the minimum update to the model weights that would resulting in the correct classification receiving a model score that is higher than all incorrect classifications by a margin proportional to the loss associated with each incorrect example.

In the context of this work, the sentence level loss associated with the incorrect translations is the difference between the evaluation metric score assigned to the target translations minus the evaluation score obtained when the learning target for the current sentence is replaced by another entry on the *n*-best list.

It is not always possible to find weights that assign a higher model score to the target translation than the alternative examples on the *n*-best list. Similarity, there may also be data points where achieving the desired separation would result in a dramatic change to the weights that would cause the model to effectively "forget" much of what it had already learned from prior training examples. To accommodate for this, slack variables are used to allow the algorithm to trade off the size of each weight update with the degree to which the learning targets are separated from the incorrect examples.

To determine the appropriate weight update, MIRA solves the quadratic programming problem given in (6.1) subject to the constraints in (6.2). This simply formalizes the description above, as the procedure attempts to make the minimal weight update $\Delta\mathbf{w}$ that separates the feature space representations of the learning target $\Psi(\mathbf{e}^r)$ from all other possible translations by a margin equal to the difference in evaluation score

between the learning target and each alternative translation $\mathbf{e}$ as given by $\Delta(\mathbf{e}^r, \mathbf{e})$.

$$\underset{\Delta\mathbf{w}}{\operatorname{argmin}} \frac{1}{2}||\Delta\mathbf{w}||^2 + C\xi \tag{6.1}$$

$$\begin{aligned}(\mathbf{w} + \Delta\mathbf{w})\left(\Psi(\mathbf{e}^r, \mathbf{f}) - \Psi(\mathbf{e}, \mathbf{f})\right) &\geq \Delta(\mathbf{e}^r, \mathbf{e}) + \xi \\ &\forall \mathbf{e} \neq \mathbf{e}^r\end{aligned} \tag{6.2}$$

The number of constraints for each training example in (6.2) scales with the number of possible translations $\mathbf{e}$ for each source sentence $\mathbf{f}$. This would result in a massive number of constraints if optimization was performed over the complete space of translations $\mathbf{e}$. However, since, similar to MERT, I perform MIRA training using n-best lists produced by the decoder, the number of constraints is only equal to the size of the n-best lists.

The quadratic programming problem above was solved for each weight update using the approach suggested in Chiang et al. (2008). The problem is converted into its dual form and the resulting dual parameters are set using a pairwise sequential minimal optimization.

6.2.3 Support Vector Machines for Interdependent Structured Output Spaces

Support vector machines for interdependent output spaces (SVMISO) are a generalization of SVMs to structured prediction tasks (Tsochantaridis et al., 2004). The approach operates by attempting to separate all learning targets from alternatives by the largest possible margin. If it is not possible to separate all alternatives from the learning targets, the algorithm makes use of a slack penalty scaled by the inverse of the loss. This formulation causes the algorithm to weight margin violations with high loss more than margin violations with lower loss.

This approach solves the quadratic programming problem given in (6.3) subject to the constraints given in (6.4). Unlike an online algorithm such as MIRA, SVMISO finds the global solution to this optimization problem for all of the data points i in a training set.

$$\underset{w}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \tag{6.3}$$

$$\mathbf{w}\Phi(\mathbf{f},\mathbf{e}_i^r) - \mathbf{w}\Phi(\mathbf{f},\mathbf{e}) \geq 1 - \frac{\xi_i}{\Delta(\mathbf{e}_i^t,\mathbf{e})} \quad \forall_i \forall \mathbf{e} \neq \mathbf{e}_i^t \tag{6.4}$$

The number of constraints for each training example in (6.4) scales with the number of possible translations $\mathbf{e}$ for each source sentence $\mathbf{f}_i$. This optimization problem is solved efficiently using a cutting-plane algorithm that incrementally adds the most violated constraints until none of the constraints are violated by more than some small convergence criterion ε. While this approach is not strictly necessary when training using only *n*-best lists, it is still used in this work in order to speed up the training procedure.

6.2.4 Max Margin Markov networks

As with both MIRA and SVMISO, max-margin Markov networks (Taskar et al., 2004) attempt to maximize the separation between the learning targets $\mathbf{e}^r$ and all other possible translations $\mathbf{e}$. Rather than taking SVMISO's approach of scaling the margin violation penalty, max-margin Markov networks are similar to MIRA in that they scale the size of the desired margin by the loss. However, the approach differs from MIRA in that instead of doing sentence-level updates, max-margin Markov networks attempt to globally optimize the model weights over an entire data set. The approach is formalized as the quadratic programming problem given in (6.5) optimized subject to the constraints given in (6.6). It attempts to separate learning targets $\mathbf{e}^r$ from all other possible translations $\mathbf{e}$ by a distance proportional to the difference in the evaluation metric score associated with the two translations $\Delta(\mathbf{e}^r,\mathbf{e})$. If this is not possible, violations of the scaled margins are penalized uniformly. Optimization is performed using the cutting plane approach described above for SVMISO.

$$\underset{w}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\sum \xi_i \tag{6.5}$$

$$\mathbf{w}\Psi(\mathbf{f},\mathbf{e}_i^r) - \mathbf{w}\Psi(\mathbf{f},\mathbf{e}) \geq \Delta(\mathbf{e}_i^r,\mathbf{e}) - \xi_i \quad \forall_i \forall \mathbf{e} \neq \mathbf{e}_i^r \tag{6.6}$$

6.2.5 Log-linear models

Log-linear models[3] assign a probability to the translation of source sentence **f** to target sentence **e** using equation (6.7). First, a linear score is computed for the translation of **f** to **e** using the model weights **w** and the features $\Psi(\mathbf{e},\mathbf{f})$. To map this linear score to a probability, it is first converted to a value between 0 and positive infinity by the natural exponential function e. This value is then normalized by dividing it by the exponential function transformed scores for all other possible translations.

$$P(\mathbf{e}_i^r|\mathbf{f}_i) = \frac{e^{\mathbf{w}\Psi(\mathbf{e}_i^r,\mathbf{f}_i)}}{\sum_{\mathbf{e}} e^{\mathbf{w}\Psi(\mathbf{e},\mathbf{f}_i)}} \tag{6.7}$$

To fit a log-linear model to a training set, its weights are set by maximizing the conditional likelihood of the desired target translations given the source sentences. This results in the convex optimization problem given in (6.8). In order to control for overfitting of the training data, the objective function is typically regularized by a quadratic penalty as seen in (6.9). Both objectives can be optimized using gradient descent based optimization techniques. In this work, optimization is performed using l-BFGS and experiments are presented for both regularized and unregularized training.

$$\underset{\mathbf{w}}{\operatorname{argmin}} - \sum_i \log P(\mathbf{e}_i^r|\mathbf{f}_i) \tag{6.8}$$

$$\underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2\sigma}||\mathbf{w}||^2 - \sum_i \log P(\mathbf{e}_i^r|\mathbf{f}_i) \tag{6.9}$$

[3] The term "log-linear" models is currently widely used in machine translation to describe any decoding model that makes use of a handful of features and score translations using a weighted linear combination of the feature values. In this work, when I use the term log-linear model, I specifically mean log-linear models trained to a conditional log-likelihood objective.

6.3 Experiments

Experiments were run using the Phrasal machine translation package (Cer et al., 2010b). During decoding I made use of a stack size of 100, set the distortion limit to 6, and retrieved 20 translation options for each unique source phrase. For each learning algorithm, I train both Chinese to English and Arabic to English models. The Chinese to English models are trained using NIST MT02, with NIST MT03 being used as dev-test and NIST MT05 being used as an evaluation set. The Arabic to English experiments use NIST MT06 for training and GALE dev07 as dev-test.

Training using all of the different learning algorithms proceeds similarly to what is typically done for MERT. First, the decoder is run using untuned weights.[4] The resulting *n*-best lists are then used to learn new model weights with the selected learning algorithm. The decoder is then run with the newly learned weights to produce another set of *n*-best lists. The new *n*-best lists are concatenated with the *n*-best lists from the prior iteration and learning is performed using the combined *n*-best lists. For all learning algorithms except MIRA, training was terminated when the change in each weight during the last iteration of learning was less than $1e-5$. For MIRA, learning was stopped after 10 epochs.

6.3.1 Arabic to English

The Arabic to English system was based on Stanford's well ranking 2009 NIST submission. The phrase-table was extracted using all of the allowed resources for the constrained Arabic to English track. Word alignment was performed using the Berkeley cross-EM aligner (Liang et al., 2006b). Phrases were extracted using the grow heuristic (Koehn et al., 2003). However, all phrases that have a $P(e|f) < 0.0001$ were thrown away in order to reduce the size of the phrase-table. From the aligned data, a hierarchical reordering model that is similar to popular lexical reordering models (Koehn et al., 2007) but that models swaps containing more than just one phrase (Galley & Manning, 2008) was also extracted. A 5-gram language model was created with the SRI language modeling toolkit (Stolcke, 2002) using all of the English material

[4] As with MERT, the weights used during the first iteration of the algorithm are a generic set of manually selected weights. I use: a weight of 1.0 for all lexicalized reordering features, linear distortion, and the language model; a weight of 0.3 for translation model scores $P_{lex}(\mathbf{e}|\mathbf{f})$ and $P_{lex}(\mathbf{f}|\mathbf{e})$; and 0.2 for $P(\mathbf{e}|\mathbf{f})$ and $P(\mathbf{f}|\mathbf{e})$. All other model weights were set to zero.

from the parallel data employed to train the phrase-table as well as Xinhua Chinese English Parallel News (LDC2002E18).[5] The resulting decoding model has 16 features that are optimized during MERT.

6.3.2 Chinese to English

For the Chinese to English system, the phrase-table was built using 1,140,693 sentence pairs sampled from the GALE Y2 training data. The Chinese sentences were word segmented using the 2008 version of Stanford Chinese Word Segmenter (Chang et al., 2008; Tseng et al., 2005). Phrases were extracted by running GIZA++ (Och & Ney, 2003) in both directions and then merging the alignments using the grow-diag-final heuristic (Koehn et al., 2003). From the merged alignments, a bidirectional lexical reordering model conditioned on the source and the target phrases (Koehn et al., 2007) was also extracted. A 5-gram language model was created with the SRI language modeling toolkit (Stolcke, 2002) and trained using the Gigaword corpus and English sentences from the parallel data. The resulting decoding model has 14 features to be trained.

6.4 Results

Table 6.1 reports results for the different learning algorithms on Chinese to English translation and table 6.2 reports on Arabic to English. The models are evaluated on BLEU and trained using learning targets selected using BLEU. For both language pairs, the results show that almost all of the learning algorithms produce performance that is very close to that of MERT. On Chinese to English, regularized log-linear training even performs 0.5 BLEU points better than MERT on the dev-test set and 0.7 BLEU points better than MERT on the test set. On Arabic to English, the regularized log-linear model marginally outperforms MERT by 0.2 BLEU points on the dev-test set.

Both globally optimized max margin techniques, SVMISO and max-margin Markov networks, achieve similar performance to MERT. However, max-margin Markov networks are able to roughly match the performance of MERT, while SVMISO seems to perform slightly worse than the MERT models on both language

[5] In order to run multiple experiments in parallel on the computers available to me, the system I use for this work differs from the Stanford 2009 NIST submission in that the Google *n*-gram language model was removed. This results in a performance drop of less than 1.0 BLEU point on the dev data.

Method	MT02 dev	MT03 dev-test	MT05 test
Log-Linear (Unreg)	30.9	31.5	29.9
Log-Linear ($\sigma = 100$)	31.8	**32.3**	**30.8**
SVMISO	30.8	30.7	29.6
Max Margin Markov	31.3	31.6	30.2
MIRA	**31.9**	31.8	30.1
MERT	**31.9**	31.7	30.1

Table 6.1: **Learning Algorithm Comparison - Chinese:** Chinese to English training, dev-test and test performance using different learning algorithms and evaluated using BLEU. Learning targets were selected using BLEU for the algorithms that require them.

pairs. The only difference between these two techniques is that max-margin Markov networks use the loss between the learning target and other translation candidates to scale the margin, while SVMISO uses the loss to scale the penalty for margin violations. These results suggest that for difficult to separate data points as is produced by phrase-based machine translation systems that make use of limited numbers of features, scaling the margin may be preferable than trying to enforce a uniform margin with a scaled penalty for margin errors.

Method	MT06 dev	GALE dev07 dev-test
Log-Linear (Unreg)	45.7	45.9
Log-Linear ($\sigma = 100$)	46.1	**46.3**
SVMISO	45.2	45.7
Max Margin Markov	**46.3**	46.2
MIRA	45.9	46.0
MERT	46.2	46.1

Table 6.2: **Learning Algorithm Comparison - Arabic:** Arabic to English training and dev-test performance using different learning algorithms and evaluated using BLEU. Learning targets were selected using BLEU for the algorithms that require them.

6.5 Analysis

These results contrast with the conclusions drawn in the original work that proposed MERT as a vastly superior approach to training. However, closer inspection of Och (2003) reveals that his results are not actually inconsistent with the results above.

In Och (2003), the baseline log-linear model is trained to learning targets selected using word error

rate (WER) (Nießen et al., 2000) rather than BLEU. Its performance was then contrasted with MERT models tuned to four different evaluation metrics: WER, PER, BLEU, and NIST. The performance of the resulting systems was then measured using each of these evaluation metrics. Each of the MERT trained systems outperformed the baseline log-linear system on the evaluation metric to which it was trained. In some cases, the performance differences were enormous. The MERT model fit to BLEU outperformed the log-linear baseline system by 5.9 BLEU points on the test data.

However, a comparison between the MERT system fit to BLEU and Och's log-linear model is not completely appropriate, since the baseline log-linear system was trained toward WER rather than BLEU. Och (2003) and more recently Cer et al. (2010c) both show that system performance can degrade significantly if the decoding model is fit to one evaluation metric then scored by a different metric.

A more appropriate comparison is between Och's baseline log-linear model and his MERT system that is fit to WER. When comparing these two models also using WER as the evaluation metric, Och (2003)'s MERT WER model only outperformed the log-linear baseline model by 0.3 WER points. Since the learning targets for the log-linear baseline model were selected using WER, this result shows that fitting a model to a specific evaluation metric just by selecting learning targets using that metric works nearly as well as fitting a model to the same metric using MERT.

Another difference between my results and those of Och (2003) concerns regularization. Och (2003) used an unregularized log-linear model for his baseline. However, regularization has been shown to be important for achieving good generalization performance in log-linear models. In my results, as in Och (2003), the unregularized log-linear model does only marginally worse than the model trained using MERT. However, once I introduce regularization, the log-linear model actually performs better than MERT.

Taken together, these results show that when the learning targets are selected using the same evaluation metric that will be used to score the system, regularized log-linear models are competitive with, and may even be superior to, MERT models.

6.6 Discussion

The significance of the results presented here can be characterized as follows. It is not necessary to use MERT in order to obtain to good performance from statistical machine translation systems. Rather, and similar to other natural language processing tasks, a variety of learning algorithms can be used with largely comparable results. While prior work has shown that the MIRA algorithm can do as well as MERT (Chiang et al., 2008; Chiang et al., 2009), this works extends this result to other popular learning algorithms.

One of the primary weaknesses of MERT is that it does not scale well to larger feature sets. All of the other learning algorithms explored in this work do scale well to larger numbers of features. Being able to use these methods should encourage more work similar to Chiang et al. (2009) and Watanabe et al. (2006) that explore richer feature sets.

6.7 Conclusion

Since its introduction, MERT has dominated training machine translation decoding models. This dominance is based on the perception that, at least for machine translation, MERT does vastly better than alternative learning algorithms. While recent work has suggested that the MIRA algorithm is competitive with MERT, there has been no thorough benchmarking of other popular learning algorithms. When I performed this benchmarking, the results show that MIRA is not the only viable alternative to MERT. Rather, the results suggest that any competitive machine learning algorithm can be used. These results should encourage future work to feel at liberty to use other learning algorithms for training machine translation decoding models. Since, unlike MERT, most other learning algorithms scale well to larger numbers of features, this will hopefully also help further encourage the general use of decoding models with richer feature sets.

Chapter 7

Conclusion and Future Work

MERT is and has long been the dominant approach for parameterizing the decoding models used for statistical machine translation. For typical machine translation systems that make use of a modest number of model features, the popular implementations of MERT are able to effectively search for the model weights that achieve the best possible evaluation score either using downhill simplex or line search methods that make use of Och (2003)'s global minimum line search. Fitting decoding models directly to evaluation metrics makes it plausible that the development of improved evaluation metrics could effortlessly lead to improved machine translation systems. As metrics get better at reproducing human judgments about translation quality, MERT will always set the decoding model weights to obtain the best possible translations according to the evaluation metric and given the limitations of the system being tuned. However, MERT does bring with it certain limitations, such as not being able to scale well up to larger numbers of features and not being able to easily make use of slower evaluation metrics.

In this work, I investigated a number of issues related to the training of statistical machine translation systems using both MERT and other alternative techniques from the mainstream machine learning community. I proposed a new search strategy for MERT based on using randomized search directions as well as a method for regularizing the MERT objective. I systematically investigated using MERT to fit different popular evaluation metrics. I demonstrated the use of a method for fitting slower evaluation metrics based on staged training, first using a faster evaluation metric like BLEU or TER and then switching to training using small n-best lists and the slower evaluation metric of interest. Finally, I contrasted the effectiveness of a number of different machine learning methods with MERT. The results of these investigations have

challenged some of the assumptions and beliefs about MERT within the machine translation community.

Significantly, I have shown that recent improvements in evaluation metrics do not seem to lead to corresponding improvements in the machine translation systems trained to these new metrics. Using MERT to train a system to a newer metric that correlates better with human judgments does result in a machine translation system that scores higher on that metric than a system trained to an older metric like BLEU. However, when the resulting system translations are given to human judges, there is no clear preference for the translations produced by systems trained to popular new evaluation metrics like TER, TERp and METEOR over those from a BLEU trained system. This shows that MERT does not necessarily allow us to effortlessly convert improvements in evaluation metrics to improvements in the quality of the machine translation systems that are tuned to them. Rather, it is proposed that improvements in an evaluation metric only produce corresponding improvements in the quality of translations produced by a system tuned to the metric to the extent that there is a correspondence between what is being measured by the evaluation metric and the structure and features of the decoding model.

The widespread usage of MERT within machine translation was largely motivated by early experimental results showing that MERT produces better systems than those trained using log-linear models. This result suggested typical machine learning algorithms did not work well for machine translation and that MERT should be used instead. While some recent work using MIRA has shown that MIRA training is as effective as MERT (Chiang et al., 2008; Chiang et al., 2009), no one has systematically revisited the effectiveness of different popular machine learning algorithms for training machine translation models. By doing so in this work, I have established that neither MIRA or MERT are special with respect to machine translation and that rather any competitive machine learning algorithm can be used to train machine translation models with roughly equal effectiveness.

The results presented earlier in Och (2003) may have led people to overestimate the effectiveness of MERT, since they compared a log-linear model trained to target translations selected using WER to the performance of models trained using MERT to a variety of other evaluation metrics. For all evaluation metrics except for WER, the models trained using MERT to whatever metric was used for evaluation did much better than the WER trained model. When evaluating model performance using WER, Och (2003)'s

results show that the log-linear model performs only slightly worse than a model trained using MERT to WER. Similar to Och (2003), the results presented in this work show that if smoothing is not used and the evaluation metric chosen to select the learning targets is the same as the one that will be ultimately used at test time, log-linear models do in fact perform only marginally worse than MERT models. However, I also show that if smoothing is used, log-linear models perform as good as or better than models trained using MERT.

Hopefully, these results will encourage future work to explore some of the potential advantages of other learning algorithms over MERT. For instance, while MERT does not scale effectively to very large numbers of model features, most other learning algorithms do. By using these other learning algorithms, research in machine translation can make use of the sort of massive feature sets that have proven so successful for other natural language processing tasks. As seen in chapter 4, MERT can also be a relatively unstable training method, with different runs producing models of significantly different model quality. Other learning algorithms may be more stable than MERT and result in more trustworthy results when contrasting different system configurations.

7.1 Future Work

It would be worthwhile to investigate learning stability using learning algorithms other than MERT. Once a set of learning targets are selected, all of the alternative learning algorithms explored in this work operate by solving convex optimization problems. This means that the inner loop during learning is not sensitive to noise introduced by the random initial starting points. While this should lead to more stable results, it would be worth investigating how significantly other sources of noise impact learning using the other algorithms. For example, the set of initial weights that are used by the decoder to generate the first *n*-best lists given to the learning algorithm may have a significant impact on the quality of the model that is ultimately learned. Also, when selecting learning targets, any alternative selections that result in the same evaluation metric score will be effectively chosen at random.[1] It is unclear to what extent different random

[1] e.g., potential learning targets that either have the same surface form but that are generated using different internal alignments or potential targets with surface form differences that are ignored by the evaluation metric such as two candidates containing two different incorrect translations of a word or phrase.

selections can cause significant differences in the resulting models. While it is expected that the alternative learning algorithms will not be too sensitive to random variations in the selection of learning targets that all receive the same evaluation score, they may be more sensitive than MERT to variations in the weights used to generate the first set of *n*-best lists. For the latter, training to explicit learning targets may cause the system to be more likely to "commit" to common errors produced by the initial weights. This may particularly be a problem if very large discriminative feature sets are used.

Prior work has demonstrated that MIRA can both produce machine translation models that are as good as those tuned using MERT and it can also be used to effectively train machine translation decoding models with large numbers of features (Watanabe et al., 2006; Chiang et al., 2008; Chiang et al., 2009). Since this work has suggested that it is actually possible to effectively train machine translation models using any state-of-the-art learning algorithm, it would be interesting to compare and contrast how well other algorithms scale up to larger numbers of features. While it is expected that the results will mirror those presented here for training traditional phrase-based models that contain a relatively small number of features, demonstrating this definitively would strongly establish that all state-of-the-art learning algorithms are truly equivalent for machine translation.

Bibliography

Agarwal, A., & Lavie, A. (2008). METEOR, M-BLEU and M-TER: evaluation metrics for high-correlation with human rankings of machine translation output. Proceedings of the ACL Third Workshop on Statistical Machine Translation (WMT) (pp. 115–118). Stroudsburg, PA, USA: Association for Computational Linguistics.

Amigó, E., Giménez, J., Gonzalo, J., & Màrquez, L. (2006). Mt evaluation: Human-like vs. human acceptable. Proceedings of the Joint 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL) (pp. 17–24). Sydney, Australia: Association for Computational Linguistics.

Arun, A., & Koehn, P. (2007). Online learning methods for discriminative training of phrase based statistical machine translation. Proceedings of the Machine Translation Summit XI. Copenhagen, Denmark: International Association for Machine Translation.

Auli, M., Lopez, A., Hoang, H., & Koehn, P. (2009). A systematic analysis of translation model search spaces. Proceedings of the EACL Fourth Workshop on Statistical Machine Translation (WMT) (pp. 224–232). Athens, Greece: Association for Computational Linguistics.

Avramidis, E., & Koehn, P. (2008). Enriching morphologically poor languages for statistical machine translation. Proceedings of the Joint 46th Annual Meeting of the Association for Computational Linguistics and the Human Language Technology Conference (HLT/ACL) (pp. 763–770). Columbus, Ohio: Association for Computational Linguistics.

Banerjee, S., & Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and Summarization (pp. 65–72). Ann Arbor, MI.

Bannard, C., & Callison-Burch, C. (2005). Paraphrasing with bilingual parallel corpora. Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL) (pp. 597–604). Stroudsburg, PA, USA: Association for Computational Linguistics.

Bar-Haim, R., Dagan, I., Greental, I., Szpektor, I., & Friedman, M. (2007). Semantic inference at the lexical-syntactic level for textual entailment recognition. Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing (pp. 131–136). Prague.

Barzilay, R., Elhadad, N., & McKeown, K. R. (2002). Inferring strategies for sentence ordering in multidocument news summarization. Journal of Artificial Intelligence Research, 17, 35–55.

Blalock, H. M. (1972). Social statistics. New York: McGraw-Hill. 2nd edition.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., & Mercer, R. L. (1993). The mathematics of statistical machine translation: parameter estimation. Computational Linguistics (pp. 263–311). Cambridge, MA, USA: MIT Press.

Bruckschlegel, T. (2005). Microbenchmarking C++, C#, and Java. C/C++ Users Journal.

Burchardt, A., Reiter, N., Thater, S., & Frank, A. (2007). A semantic approach to textual entailment: System evaluation and task analysis. Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing (pp. 10–15). Prague, Czech Republic.

Callison-Burch, C. (2009). Fast, cheap, and creative: Evaluating translation quality using Amazon's Mechanical Turk. Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 286–295). Singapore: Association for Computational Linguistics.

Callison-Burch, C., Fordyce, C., Koehn, P., Monz, C., & Schroeder, J. (2008). Further meta-evaluation of machine translation. Proceedings of the ACL Third Workshop on Statistical Machine Translation (WMT) (pp. 70–106). Columbus, OH.

Callison-Burch, C., Osborne, M., & Koehn, P. (2006). Re-evaluating the role of BLEU in machine translation research. Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL) (pp. 249–256). Trento, Italy.

Carpuat, M., & Wu, D. (2007). Improving statistical machine translation using word sense disambiguation. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL) (pp. 61–72). Prague, Czech Republic: Association for Computational Linguistics.

Cer, D., de Marneffe, M.-C., Jurafsky, D., & Manning, C. D. (2010a). Parsing to stanford dependencies: Trade-offs between speed and accuracy. Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010). European Language Resources Association.

Cer, D., Galley, M., Jurafsky, D., & Manning, C. D. (2010b). Phrasal: A statistical machine translation toolkit for exploring new model features. Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics Demonstration Session (HLT/NAACL) (pp. 9–12). Los Angeles, California: Association for Computational Linguistics.

Cer, D., Manning, C. D., & Jurafsky, D. (2010c). The best lexical metric for phrase-based statistical MT system optimization. Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 555–563). Stroudsburg, PA, USA: Association for Computational Linguistics.

Chan, Y. S., & Ng, H. T. (2009). MaxSim: performance and effects of translation fluency. Machine Translation, 23, 157–168.

Chang, P.-C., Galley, M., & Manning, C. D. (2008). Optimizing chinese word segmentation for machine translation performance. Proceedings of the ACL Third Workshop on Statistical Machine Translation (WMT) (pp. 224–232). Stroudsburg, PA, USA: Association for Computational Linguistics.

Chang, P.-C., Tseng, H., Jurafsky, D., & Manning, C. D. (2009). Discriminative reordering with chinese grammatical relations features. Proceedings of the NAACL/HTL Third Workshop on Syntax and Structure

in Statistical Translation (SSST) (pp. 51–59). Morristown, NJ, USA: Association for Computational Linguistics.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL) (pp. 263–270). Stroudsburg, PA, USA: Association for Computational Linguistics.

Chiang, D., Knight, K., & Wang, W. (2009). 11,001 new features for statistical machine translation. Proceedings of the Joint 2009 Annual Meeting of the North American Chapter of the Association for Computational Linguistics and the Human Language Technologies Conference (HLT/NAACL) (pp. 218–226). Stroudsburg, PA, USA: Association for Computational Linguistics.

Chiang, D., Lopez, A., Madnani, N., Monz, C., Resnik, P., & Subotin, M. (2005). The hiero machine translation system: extensions, evaluation, and analysis. Proceedings of the Joint conference on Human Language Technology and the conference on Empirical Methods in Natural Language Processing (HLT/EMNLP) (pp. 779–786). Morristown, NJ, USA: Association for Computational Linguistics.

Chiang, D., Marton, Y., & Resnik, P. (2008). Online large-margin training of syntactic and structural translation features. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 224–233). Stroudsburg, PA, USA: Association for Computational Linguistics.

Cleveland, W. S., & Devlin, S. J. (1988). Locally weighted regression: An approach to regression analysis by local fitting. Journal of the American Statistical Association, 83, 596–610.

Cohen, W. W., Schapire, R. E., & Singer, Y. (1999). Learning to order things. Journal of Artificial Intelligence Research, 10, 243–270.

Crammer, K., & Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. Journal of Machine Learning Research, 3, 951–991.

Dagan, I., Glickman, O., Gliozzo, A., Marmorshtein, E., & Strapparava, C. (2006). Direct word sense matching for lexical substitution. Proceedings of the Joint 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL) (pp. 449–456). Stroudsburg, PA, USA: Association for Computational Linguistics.

Dagan, I., Glickman, O., & Magnini, B. (2005). The PASCAL recognising textual entailment challenge. Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment. Southampton, UK.

de Marneffe, M.-C., Grenager, T., MacCartney, B., Cer, D., Ramage, D., Kiddon, C., & Manning, C. D. (2007). Aligning semantic graphs for textual inference and machine reading. Proceedings of the AAAI Spring Symposium. Stanford, CA.

de Marneffe, M.-C., MacCartney, B., & Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006). Genoa, Italy: European Language Resources Association.

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. Proceedings of the second international conference on Human Language Technology Research (HLT) (pp. 138–145). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Galley, M., Graehl, J., Knight, K., Marcu, D., DeNeefe, S., Wang, W., & Thayer, I. (2006). Scalable inference and training of context-rich syntactic translation models. Proceedings of the Joint 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL) (pp. 961–968). Stroudsburg, PA, USA: Association for Computational Linguistics.

Galley, M., Green, S., Cer, D., Chang, P.-C., & Manning, C. D. (2009). Stanford University's Arabic-to-English statistical machine translation system for the 2009 NIST evaluation. NIST Open Machine Translation Evaluation Meeting.

Galley, M., & Manning, C. D. (2008). A simple and effective hierarchical phrase reordering model. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 848–856). Stroudsburg, PA, USA: Association for Computational Linguistics.

Galley, M., & Manning, C. D. (2009). Quadratic-time dependency parsing for machine translation. Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL/IJCNLP) (pp. 773–781). Suntec, Singapore: Association for Computational Linguistics.

Galley, M., & Manning, C. D. (2010). Accurate non-hierarchical phrase-based translation. Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL) (pp. 966–974). Stroudsburg, PA, USA: Association for Computational Linguistics.

Giménez, J., & Márquez, L. (2008). Heterogeneous automatic MT evaluation through non-parametric metric combinations. Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNLP) (pp. 319–326). Hyderabad, India.

Giménez, J., & Màrquez, L. (2008). A smorgasbord of features for automatic MT evaluation. Proceedings of the ACL Third Workshop on Statistical Machine Translation (WMT) (pp. 195–198). Columbus, Ohio: Association for Computational Linguistics.

Gimpel, K., & Smith, N. A. (2010). Softmax-margin CRFs: training log-linear models with cost functions. Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 733–736). Stroudsburg, PA, USA: Association for Computational Linguistics.

Glickman, O., & Dagan, I. (2005). Web based probabilistic textual entailment. Proceedings of the 1st Pascal Challenge Workshop (pp. 33–36).

Green, S., Galley, M., & Manning, C. D. (2010). Improved models of distortion cost for statistical machine translation. Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 867–875). Stroudsburg, PA, USA: Association for Computational Linguistics.

Harabagiu, S., & Hickl, A. (2006). Methods for using textual entailment in open-domain question answering. Proceedings of the Joint 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL) (pp. 905–912). Stroudsburg, PA, USA: Association for Computational Linguistics.

He, Y., & Way, A. (2010). Metric and reference factors in minimum error rate training. Machine Translation, 24, 27–38.

Hovy, E., Lin, C.-Y., Zhou, L., & Fukumoto, J. (2006). Automated summarization evaluation with basic elements. Proceedings of the Fifth Conference on Language Resources and Evaluation (LREC). Genoa, Italy: European Language Resources Association.

Jijkoun, V., & de Rijke, M. (2005). Recognizing textual entailment: Is word similarity enough? First PASCAL Machine Learning Challenges Workshop (pp. 449–460). Southampton, UK.

Joachims, T. (2005). A support vector method for multivariate performance measures. Proceedings of the 22nd international conference on Machine learning (ICML) (pp. 377–384). New York, NY, USA: ACM.

Kauchak, D., & Barzilay, R. (2006). Paraphrasing for automatic evaluation. Proceedings of Human Language Technologies: The 2006 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 455–462).

Koehn, P. (2004). Pharaoh: A beam search decoder for phrase-based statistical machine translation models. Proceedings of the Sixth Conference of the Association for Machine Translation in the Americas (AMTA) (pp. 115–124). Stroudsburg, PA: Association for Machine Translation in the Americas.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. Proceedings of the Machine Translation Summit X. Phuket, Thailand.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., & Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Demo and Poster Sessions (ACL) (pp. 177–180). Stroudsburg, PA: Association for Computational Linguistics.

Koehn, P., Och, F. J., & Marcu, D. (2003). Statistical phrase-based translation. Proceedings of Human Language Technologies: The 2003 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 48–54). Stroudsburg, PA, USA: Association for Computational Linguistics.

Koo, T., & Collins, M. (2005). Hidden-variable models for discriminative reranking. Proceedings of the Joint conference on Human Language Technology and the conference on Empirical Methods in Natural Language Processing (HLT/EMNLP) (pp. 507–514). Stroudsburg, PA, USA: Association for Computational Linguistics.

Lavie, A., & Denkowski, M. (2009). The METEOR metric for automatic evaluation of machine translation. Machine Translation, 23, 105–115.

Levinson, S. C. (1983). Pragmatics, chapter Conversational implicature. Cambridge, UK: Cambridge University Press.

Li, Z., Callison-Burch, C., Dyer, C., Khudanpur, S., Schwartz, L., Thornton, W., Weese, J., & Zaidan, O. (2009). Joshua: An open source toolkit for parsing-based machine translation. Proceedings of the ACL joint fifth workshop on statistical machine translation and NIST Metrics for Machine Translation (WMT/MetricsMATR) (pp. 135–139).

Liang, P., Bouchard-Côté, A., Klein, D., & Taskar, B. (2006a). An end-to-end discriminative approach to machine translation. Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (COLING/ACL) (pp. 761–768). Stroudsburg, PA, USA: Association for Computational Linguistics.

Liang, P., Taskar, B., & Klein, D. (2006b). Alignment by agreement. Proceedings of Human Language Technologies: The 2006 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 104–111). Stroudsburg, PA, USA: Association for Computational Linguistics.

Lin, C.-Y., & Och, F. J. (2004). ORANGE: a method for evaluating automatic evaluation metrics for machine translation. Proceedings of the 20th International Conference on Computational Linguistics (COLING) (pp. 501–507). Geneva, Switzerland: COLING.

Lita, L. V., Ittycheriah, A., Roukos, S., & Kambhatla, N. (2003). tRuEcasIng. Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL) (pp. 152–159). Stroudsburg, PA, USA: Association for Computational Linguistics.

Liu, D., & Gildea, D. (2005). Syntactic features for evaluation of machine translation. Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization (pp. 25–32). Stroudsburg, PA, USA: Association for Computational Linguistics.

MacCartney, B., Grenager, T., de Marneffe, M.-C., Cer, D., & Manning, C. D. (2006). Learning to recognize features of valid textual entailments. Proceedings of the North American Association of Computational Linguistics (NAACL). Stroudsburg, PA: Association for Computational Linguistics.

MacCartney, B., & Manning, C. D. (2008). Modeling semantic containment and exclusion in natural language inference. Proceedings of the 22nd International Conference on Computational Linguistics (COLING) (pp. 521–528). Manchester, UK: COLING.

Marsi, E., Krahmer, E., & Bosma, W. (2007). Dependency-based paraphrasing for recognizing textual entailment. Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing (pp. 83–88). Stroudsburg, PA, USA: Association for Computational Linguistics.

Mauser, A., Hasan, S., Parlikar, A., & Vogel, S. (in press). Handbook of natural language processing and machine translation: DARPA Global Autonomous Language Exploitation, chapter Machine Translation Evaluation, section Effect of MT Evaluation Measures on Optimization. DARPA/Springer. eds. Joseph Olive, John McCary, Paul Dietrich and Caitlin Christianson.

Monz, C., & de Rijke, M. (2001). Light-weight entailment checking for computational semantics. Proceedings of the IJCAR third workshop on inference in computational semantics (ICoS-3). Stroudsburg, PA, USA: Association for Computational Linguistics.

Nießen, S., Och, F. J., & Ney, H. (2000). An evaluation tool for machine translation: Fast evaluation for MT research. Proceeding of the 2nd International Conference on Language Resources and Evaluation (LREC) (pp. 39–45). Athens, Greece: European Language Resources Association.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL) (pp. 160–167). Stroudsburg, PA: Association for Computational Linguistics.

Och, F. J., Gildea, D., Khudanpur, S., Sarkar, A., Yamada, K., Fraser, A., Kumar, S., Shen, L., Smith, D., Eng, K., Jain, V., Jin, Z., & Radev, D. (2004). A smorgasbord of features for statistical machine translation. Proceedings of Human Language Technologies: The 2004 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 161–168). Stroudsburg, PA: Association for Computational Linguistics.

Och, F. J., & Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL) (pp. 295–302). Stroudsburg, PA: Association for Computational Linguistics.

Och, F. J., & Ney, H. (2003). A systematic comparison of various statistical alignment models. Computational Linguistics, 29, 19–51.

Och, F. J., & Ney, H. (2004). The alignment template approach to statistical machine translation. Computational Linguistics, 30, 417–449.

Olteanu, M., Davis, C., Volosen, I., & Moldovan, D. (2006). Phramer: an open source statistical phrase-based translator. Proceedings of the NAACL Workshop on Statistical Machine Translation (WMT) (pp. 146–149). Stroudsburg, PA: Association for Computational Linguistics.

Owczarzak, K., van Genabith, J., & Way, A. (2007). Dependency-based automatic evaluation for machine translation. Proceedings of the NAACL/HTL/AMTA Workshop on Syntax and Structure in Statistical Translation (SSST) (pp. 80–87). Stroudsburg, PA: Association for Computational Linguistics.

Padó, S., Galley, M., Jurafsky, D., & Manning, C. (2009a). Robust machine translation evaluation with entailment features. Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL/IJCNLP) (pp. 297–305). Stroudsburg, PA: Association for Computational Linguistics.

Padó, S., Galley, M., Jurafsky, D., & Manning., C. (2009b). Textual entailment features for machine translation evaluation. Proceedings of the EACL Workshop on Statistical Machine Translation (WMT). Stroudsburg, PA: Association for Computational Linguistics.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: a method for automatic evaluation of machine translation. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL) (pp. 311–318). Stroudsburg, PA: Association for Computational Linguistics.

Phipps, G. (1999). Comparing observed bug and productivity rates for Java and C++. Software Practice & Experience, 29, 345–358.

Platt, J. C. (1999). Using analytic QP and sparseness to speed training of support vector machines. Proceedings of the 1998 conference on advances in neural information processing systems II (NIPS) (pp. 557–563). Cambridge, MA: MIT Press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical Recipes: The Art of Scientific Computing. Cambridge University Press. 3rd edition.

Przybocki, K. P. M., & Bronsart, S. (2008). Translation adequacy and preference evaluation tool (TAP-ET). Proceedings of the Sixth International Language Resources and Evaluation (LREC). Marrakech, Morocco: European Language Resources Association.

Przybocki, M., Peterson, K., & Bronsart, S. (2008). Official results of the "Metrics for MAchine TRanslation Challenge (MetricsMATR08) (Technical Report). NIST, http://nist.gov/speech/tests/metricsmatr/2008/results/.

Riezler, S., & Maxwell, J. T. (2005). On some pitfalls in automatic evaluation and significance testing for MT. Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization (pp. 57–64). Stroudsburg, PA: Association for Computational Linguistics.

Roth, D., & Sammons, M. (2007). Semantic and logical inference model for textual entailment. Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing (pp. 107–112). Stroudsburg, PA: Association for Computational Linguistics.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J. C., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. Neural Computation, 13, 1443–1471.

Sekine, S., Inui, K., Dagan, I., Dolan, B., Giampiccolo, D., & Magnini, B. (Eds.). (2007). Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing. Association for Computational Linguistics.

Shen, L., Sarkar, A., & Och, F. J. (2004). Discriminative reranking for machine translation. Proceedings of Human Language Technologies: The 2004 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 177–184). Stroudsburg, PA: Association for Computational Linguistics.

Smith, D. A., & Eisner, J. (2006). Minimum risk annealing for training log-linear models. Proceedings of the Joint 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL) (pp. 787–794). Stroudsburg, PA: Association for Computational Linguistics.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., & Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. Proceedings AMTA (pp. 223–231). Stroudsburg, PA: Association for Machine Translation in the Americas.

Snover, M., Madnani, N., Dorr, B., & Schwartz, R. (2008). TERp system description. Proceedings of the AMTA workshop NIST Metrics for Machine Translation (MetricsMATR). Stroudsburg, PA: Association for Machine Translation in the Americas.

Snover, M., Madnani, N., Dorr, B. J., & Schwartz, R. (2009). Fluency, adequacy, or HTER?: exploring different human judgments with a tunable MT metric. Proceedings of the EACL Fourth Workshop on Statistical Machine Translation (WMT) (pp. 259–268). Stroudsburg, PA: Association for Computational Linguistics.

Snow, R., O'Connor, B., Jurafsky, D., & Ng, A. Y. (2008). Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 254–263). Stroudsburg, PA: Association for Computational Linguistics.

Stolcke, A. (2002). SRILM-An extensible language modeling toolkit. Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP) (pp. 257–286). Baixas, France: International Speech Communication Association.

Taskar, B., Guestrin, C., & Koller, D. (2004). Max-margin markov networks. Proceedings of Advances in Neural Information Processing Systems 16 (NIPS). Cambridge, MA: MIT Press.

Tillmann, C. (2004). A unigram orientation model for statistical machine translation. Proceedings of Human Language Technologies: The 2004 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (pp. 101–104). Stroudsburg, PA: Association for Computational Linguistics.

Tillmann, C., Vogel, S., Ney, H., Zubiaga, A., & Sawaf, H. (1997). Accelerated DP based search for statistical translation. Proceedings of the 5th European Conference on Speech Communication and Technology (EUROSPEECH) (pp. 2667–2670). Baixas, France: International Speech Communication Association.

Tillmann, C., & Zhang, T. (2005). A localized prediction model for statistical machine translation. Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL) (pp. 557–564). Stroudsburg, PA: Association for Computational Linguistics.

Tillmann, C., & Zhang, T. (2006). A discriminative global training algorithm for statistical MT. Proceedings of the Joint 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL) (pp. 721–728). Stroudsburg, PA: Association for Computational Linguistics.

Tseng, H., Chang, P., Andrew, G., Jurafsky, D., & Manning, C. (2005). A conditional random field word segmenter for sighan bakeoff 2005. Proceedings of the IJCNLP fourth SIGHAN Workshop on Chinese Language Processing. Stroudsburg, PA: Association for Computational Linguistics.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. Proceedings of the twenty-first International Conference on Machine Learning (ICML) (pp. 104–112). New York, NY: ACM.

Vishwanathan, S. V. N., Schraudolph, N. N., Schmidt, M. W., & Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient methods. Proceedings of the 23rd International Conference on Machine Learning (ICML) (pp. 969–976). New York, NY, USA: ACM.

Watanabe, T., Suzuki, J., Tsukada, H., & Isozaki, H. (2006). NTT statistical machine translation for IWSLT 2006. Proceedings of the third International Workshop on Spoken Language Translation (IWSLT). Allschwil, Switzerland: European Association for Machine Translation.

Watanabe, T., Suzuki, J., Tsukada, H., & Isozaki, H. (2007). Online large-margin training for statistical machine translation. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL) (pp. 764–773). Stroudsburg, PA: Association for Computational Linguistics.

Yamada, K., & Knight, K. (2001). A syntax-based statistical translation model. Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (ACL) (pp. 523–530). Stroudsburg, PA, USA: Association for Computational Linguistics.

Zaidan, O. F. (2009). Z-MERT: A fully configurable open source tool for minimum error rate training of machine translation systems. The Prague Bulletin of Mathematical Linguistics, 91, 79–88.

Zaidan, O. F., & Callison-Burch, C. (2009). Feasibility of human-in-the-loop minimum error rate training. Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 52–61). Stroudsburg, PA: Association for Computational Linguistics.

Zanzotto, F. M., Pennacchiotti, M., & Moschitti, A. (2007). Shallow semantic in fast textual entailment rule learners. Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing (pp. 72–77). Stroudsburg, PA: Association for Computational Linguistics.

Zens, R., Hasan, S., & Ney, H. (2007). A systematic comparison of training criteria for statistical machine translation. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL) (pp. 524–532). Stroudsburg, PA: Association for Computational Linguistics.

Zens, R., & Ney, H. (2006). Discriminative reordering models for statistical machine translation. Proceedings of the NAACL Workshop on Statistical Machine Translation (WMT) (pp. 55–63). Stroudsburg, PA: Association for Computational Linguistics.

Zhang, Y., & Vogel, S. (2004). Measuring confidence intervals for the machine translation evaluation metrics. Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-2004) (pp. 4–6). Stroudsburg, PA: Association for Machine Translation in the Americas.

Zhang, Y., Vogel, S., & Waibel, A. (2004). Interpreting BLEU/NIST scores: How much improvement do we need to have a better system. Proceedings of the fourth International Conference on Language Resources and Evaluation (LREC) (pp. 2051–2054). Paris France: European Language Resources Association.

Zhao, B., & Chen, S. (2009). A simplex armijo downhill algorithm for optimizing statistical machine translation decoding parameters. Proceedings of the Joint 2009 Annual Meeting of the North American Chapter of the Association for Computational Linguistics and the Human Language Technologies Conference (HLT/NAACL) (pp. 21–24). Stroudsburg, PA: Association for Computational Linguistics.

Zhou, L., Lin, C.-Y., & Hovy, E. (2006). Re-evaluating machine translation results with paraphrase support. Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 77–84). Stroudsburg, PA, USA: Association for Computational Linguistics.