Optimization for Statistical Machine Translation: A Survey

Graham Neubig*
Graduate School of Information Science
Nara Institute of Science and Technology

Taro Watanabe**
Google Inc.

In statistical machine translation (SMT), the optimization of the system parameters to maximize translation accuracy is now a fundamental part of virtually all modern systems. In this article, we survey 12 years of research on optimization for SMT, from the seminal work on discriminative models (Och and Ney 2002) and minimum error rate training (Och 2003), to the most recent advances. Starting with a brief introduction to the fundamentals of SMT systems, we follow by covering a wide variety of optimization algorithms for use in both batch and online optimization. Specifically, we discuss losses based on direct error minimization, maximum likelihood, maximum margin, risk minimization, ranking, and more, along with the appropriate methods for minimizing these losses. We also cover recent topics, including large-scale optimization, nonlinear models, domain-dependent optimization, and the effect of MT evaluation measures or search on optimization. Finally, we discuss the current state of affairs in MT optimization, and point out some unresolved problems that will likely be the target of further research in optimization for MT.

1. Introduction

Machine translation (MT) has long been both one of the most promising applications of natural language processing technology and one of the most elusive. However, over approximately the past decade, huge gains in translation accuracy have been achieved (Graham et al. 2014), and commercial systems deployed for hundreds of language pairs are being used by hundreds of millions of users. There are many reasons for these advances in the accuracy and coverage of MT, but among them two particularly stand out: statistical machine translation (SMT) techniques that make it possible to learn statistical models from data, and massive increases in the amount of data available to learn SMT models.

* 8916-5 Takayama-cho, Ikoma, Nara, Japan. E-mail: neubig@is.naist.jp.
** 6-10-1 Roppongi, Minato-ku, Tokyo, Japan. E-mail: tarow@google.com.
This work was mostly done while the second author was affiliated with the National Institute of Information and Communications Technology, 3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0289, Japan.

Submission received: 3 June 2014; revised version received: 18 March 2015; accepted for publication: 11 October 2015.

doi:10.1162/COLI_a_00241

Within the SMT framework, there have been two revolutions in the way we mathematically model the translation process. The first was the pioneering work of Brown et al. (1993), who proposed the idea of SMT, and described methods for estimation of the parameters used in translation. In that work, the parameters of a word-based generative translation model were optimized to maximize the conditional likelihood of the training corpus. The second major advance in SMT is the discriminative training framework proposed by Och and Ney (2002) and Och (2003), who propose log-linear models for MT, optimized to maximize either the probability of getting the correct sentence from a *k*-best list of candidates, or to directly achieve the highest accuracy over the entire corpus. By describing the scoring function for MT as a flexibly parameterizable log-linear model, and describing discriminative algorithms to optimize these parameters, it became possible to think of MT like many other **structured prediction** problems, such as POS tagging or parsing (Collins 2002).

However, within the general framework of structured prediction, MT stands apart in many ways, and as a result requires a number of unique design decisions not necessary in other frameworks (as summarized in Table 1). The first is the **search space** that must be considered. The search space in MT is generally too large to expand exhaustively, so it is necessary to decide which subset of all the possible hypotheses should be used in optimization. In addition, the **evaluation** of MT accuracy is not straightforward, with automatic evaluation measures for MT still being researched to this day. From the optimization perspective, even once we have chosen an automatic evaluation metric, it is not necessarily the case that it can be decomposed for straightforward integration with structured learning algorithms. Given this evaluation measure, it is necessary to incorporate it into a **loss function** to target. The loss function should be closely related to the final evaluation objective, while allowing for the use of efficient optimization algorithms. Finally, it is necessary to choose an **optimization algorithm**. In many cases it is possible to choose a standard algorithm from other fields, but there are also algorithms that have been tailored towards the unique challenges posed by MT.

Table 1
A road map of the various elements that affect MT optimization.

Which Loss Functions?	Which Optimization Algorithm?
Error (§3.1)	Minimum Error Rate Training (§5.1)
Softmax (§3.2)	Gradient-based Methods (§5.2, §6.5)
Risk (§3.3)	Margin-based Methods (§5.3)
Margin, Perceptron (§3.4)	Linear Regression (§5.4)
Ranking (§3.5)	Perceptron (§6.2)
Minimum Squared Error (§3.6)	MIRA (§6.3)
	AROW (§6.4)
Which Evaluation Measure?	**Which Hypotheses to Target?**
Corpus-level, Sentence Level (§2.5)	*k*-best vs. Lattice vs. Forest (§2.4)
BLEU and Approximations (§2.5.1, §2.5.2)	Merged *k*-bests (§5)
Other Measures (§8.3)	Forced Decoding (§2.4), Oracles (§4)

Other Topics:
Large Data Sets (§7), Non-linear Models (§8.1),
Domain Adaptation (§8.2), Search and Optimization (§8.4)

In this article, we survey the state of the art in machine translation optimization in a comprehensive and systematic fashion, covering a wide variety of topics, with a unified set of terminology. In Section 2, we first provide definitions of the problem of machine translation, describe briefly how models are built, how features are defined, and how translations are evaluated, and finally define the optimization setting. In Section 3, we next describe a variety of loss functions that have been targeted in machine translation optimization. In Section 4, we explain the selection of oracle translations, a non-trivial process that directly affects the optimization results. In Section 5, we describe batch optimization algorithms, starting with the popular minimum error rate training, and continuing with other approaches using likelihood, margin, rank loss, or risk as objectives. In Section 6, we describe online learning algorithms, first explaining the relationship between corpus-level optimization and sentence-level optimization, and then moving on to algorithms based on perceptron, margin, or likelihood-based objectives. In Section 7, we describe the recent advances in scaling training of MT systems up to large amounts of data through parallel computing, and in Section 8, we cover a number of other topics in MT optimization such as non-linear models, domain adaptation, and the relationship between MT evaluation and optimization. Finally, we conclude in Section 9, overviewing the methods described, making a brief note about which methods see the most use in actual systems, and outlining some of the unsolved problems in the optimization of MT systems.

2. Machine Translation Preliminaries and Definitions

Before delving into the details of actual optimization algorithms, we first introduce preliminaries and definitions regarding MT in general and the MT optimization problem in particular. We focus mainly on the aspects of MT that are relevant to optimization, and readers may refer to Koehn (2010) or Lopez (2008) for more details about MT in general.

2.1 Machine Translation

Machine translation is the problem of automatically translating from one natural language to another. Formally, we define this problem by specifying $\mathcal{F}$ to be the collection of all source sentences to be translated, $\boldsymbol{f} \in \mathcal{F}$ as one of the sentences, and $\mathcal{E}(\boldsymbol{f})$ as the collection of all possible target language sentences that can be obtained by translating $\boldsymbol{f}$. Machine translation systems perform this translation process by dividing the translation of a full sentence into the translation and recombination of smaller parts, which are represented as **hidden variables**, which together form a **derivation**.

For example, in phrase-based translation (Koehn, Och, and Marcu 2003), the hidden variables will be the alignment between the phrases of the source and target sentences, and in tree-based translation models (Yamada and Knight 2001; Chiang 2007), the hidden variables will represent the latent tree structure used to generate the translation. We will define $\mathcal{D}(\boldsymbol{f})$ to be the space of possible derivations that can be acquired from source sentence $\boldsymbol{f}$, and $\boldsymbol{d} \in \mathcal{D}(\boldsymbol{f})$ to be one of those derivations. Any particular derivation $\boldsymbol{d}$ will correspond to exactly one $\boldsymbol{e} \in \mathcal{E}(\boldsymbol{f})$, although the opposite is not true (the derivation uniquely determines the translation, but there can be multiple derivations corresponding to a particular translation). We also define tuple $\langle \boldsymbol{e}, \boldsymbol{d} \rangle$ consisting of a target sentence and its corresponding derivation, and $\mathcal{T}(\boldsymbol{f}) \subseteq \mathcal{E}(\boldsymbol{f}) \times \mathcal{D}(\boldsymbol{f})$ as the set of all of these tuples.

Because the set of all possible translations $\mathcal{E}(\boldsymbol{f})$ will contain both good and bad translations, it is necessary to have a method to identify and output the good

translations. In order to do so, in machine translation it is common to define a **linear model** that determines the score of each translation candidate. In this linear model we first define an M-dimensional **feature vector** for each output and its derivation as $\boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}) : \mathcal{F} \times \mathcal{E} \times \mathcal{D} \rightarrow \mathbb{R}^M$. For each feature, we also define a corresponding weight, resulting in an M-dimensional **weight vector** $\boldsymbol{w} \in \mathbb{R}^M$. Based on these feature and weight vectors, we proceed to define the problem of selecting the best $\langle \boldsymbol{e}, \boldsymbol{d} \rangle$ as the following maximization problem

$$\langle \hat{\boldsymbol{e}}, \hat{\boldsymbol{d}} \rangle = \underset{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \mathcal{T}(\boldsymbol{f})}{\arg\max}\ \boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}) \tag{1}$$

where the dot product of the parameters and features is equivalent to the score assigned to a particular translation.

The **optimization** problem that we will be surveying in this article is generally concerned with finding the most effective weight vector $\boldsymbol{w}$ from the set of possible weight vectors $\mathbb{R}^M$.[1] Optimization is also widely called **tuning** in the SMT literature. In addition, because of the exponentially large number of possible translations in $\mathcal{E}(\boldsymbol{f})$ that must be considered, it is necessary to take advantage of the problem structure, making MT optimization an instance of **structured learning**.

2.2 Model Construction

The first step of creating a machine translation system is model construction, in which **translation models** (TMs) are extracted from a large parallel corpus. The TM is usually created by first aligning the parallel text (Och and Ney 2003), using this text to extract multi-word phrase pairs or synchronous grammar rules (Koehn, Och, and Marcu 2003; Chiang 2007), and scoring these rules according to several features explained in more detail in Section 2.3. The construction of the TM is generally performed first in a manner that does not directly consider the optimization of translation accuracy, followed by an optimization step that explicitly considers the accuracy achieved by the system.[2] In this survey, we focus on the optimization step, and thus do not cover elements of model construction that do not directly optimize an objective function related to translation accuracy, but interested readers can reference Koehn (2010) for more details.

In the context of this article, however, the TM is particularly important in the role it plays in defining our derivation space $\mathcal{D}(\boldsymbol{f})$. For example, in the case of phrase-based translation, only phrase pairs included in the TM will be expanded during the process of searching for the best translation (explained in Section 2.4).

This has major implications from the point of view of optimization, the most important of which being that we must use separate data for training the TM and optimizing the parameters $\boldsymbol{w}$. The reason for this lies in the fact that the TM is constructed in such a way that allows it to "memorize" long multi-word phrases included in the training data. Using the same data to train the model parameters will result in **overfitting**, learning parameters that heavily favor using these memorized multi-word phrases, which will not be present in a separate test set.

1 It should be noted that although most work on MT optimization is concerned with linear models (and thus we will spend the majority of this article discussing optimization of these models), optimization using non-linear models is also possible, and is discussed in Section 8.1.

2 It should also be noted there have been a few recent attempts to jointly perform rule extraction and optimization, doing away with this two-step process (Xiao and Xiong 2013).

The traditional way to solve this problem is to train the TM on a large parallel corpus on the order of hundreds of thousands to tens of millions of sentences, then perform optimization of parameters on a separate set of data consisting of around one thousand sentences, often called the **development set**. When learning the weights for larger feature sets, however, a smaller development set is often not sufficient, and it is common to perform **cross-validation**, holding out some larger portion of the training set for parameter optimization. It is also possible to perform **leaving-one-out** training, where counts of rules extracted from a particular sentence are subtracted from the model before translating the sentence (Wuebker, Mauser, and Ney 2010).

2.3 Features for Machine Translation

Given this overall formulation of MT, the features $\boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d})$ that we choose to use to represent each translation hypothesis are of great importance. In particular, with regard to optimization, there are two important distinctions between types of features: local vs. non-local, and dense vs. sparse.

With regard to the first distinction, **local features**, such as phrase translation probabilities, do not require additional contexts from other partial derivations, and they are computed independently from one another. On the other hand, when features for a particular phrase pair or synchronous rule cannot be computed independently from other pairs, they are called **non-local features**. This distinction is important, as local features will not result in an increase in the size of the search space, whereas non-local features have the potential to make search more difficult.

The second distinction is between **dense features**, which define a small number of highly informative feature functions, and **sparse features**, which define a large number of less informative feature functions. Dense features are generally easier to optimize, both from a computational point of view because the smaller number of features reduces computational and memory requirements, and because the smaller number of parameters reduces the risk of overfitting. On the other hand, sparse features allow for more flexibility, as their parameters can be directly optimized to increase translation accuracy, so if optimization is performed well they have the potential to greatly increase translation accuracy. The remainder of this section describes some of the widely used features in more detail.

2.3.1 Dense Features. Dense features, which are generally continuously valued and present in nearly all translation hypotheses, are used in the majority of machine translation systems. The most fundamental set of dense features are phrase/rule **translation probabilities** or **relative frequencies** in which the log of sentence-wise probability distributions $p(\boldsymbol{f}|\boldsymbol{e})$ and $p(\boldsymbol{e}|\boldsymbol{f})$, are split into the sum of phrase or rule log probabilities

$$h_\Phi(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}) = \sum_{\langle \alpha, \beta \rangle \in \boldsymbol{d}} \log p_\Phi(\alpha|\beta), \qquad h_{\Phi'}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}) = \textstyle\sum_{\langle \alpha, \beta \rangle \in \boldsymbol{d}} \log p_{\Phi'}(\beta|\alpha) \tag{2}$$

Here α and β are the source and target sides of a phrase pair or rule. These features are estimated using counts of each phrase derived from the training corpus as follows:

$$p_\Phi(\alpha|\beta) = \frac{\text{count}(\alpha, \beta)}{\sum_{\alpha'} \text{count}(\alpha', \beta)}, \qquad p_{\Phi'}(\beta|\alpha) = \frac{\text{count}(\alpha, \beta)}{\sum_{\beta'} \text{count}(\alpha, \beta')} \tag{3}$$

In addition, it is also common to use **lexical weighting**, which estimates parameters for each phrase pair or rule by further decomposing them into word-wise probabilities (Koehn, Och, and Marcu 2003). This helps more accurately estimate the reliability of phrase pairs or rules that have low counts. It should be noted that all of these features can be calculated directly from the rules themselves, and are thus local features.

Another set of important features are language models (LMs), which capture the fluency of translation $\boldsymbol{e}$, and are usually modeled by n-grams

$$h_{\text{lm}}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}) = \sum_{i=1}^{|\boldsymbol{e}|} \log p_{\text{lm}}(e_i | e_{i-n+1}^{i-1}) \tag{4}$$

Note that the n-gram LM is computed over $\boldsymbol{e}$ regardless of the boundaries of phrase pairs or rules in the derivation, and is thus a non-local feature.

The n-gram language model assigns higher penalties for longer translations, and it is common to add a **word penalty feature** that measures the length of translation $\boldsymbol{e}$ to compensate for this. Similarly, **phrase penalty** or **rule penalty** features express the trade-off between longer or shorter derivations. There exist other features that are dependent on the underlying MT system model. Phrase-based MT heavily relies on the **distortion probabilities** that are computed by the distance on the source side of target-adjacent phrase pairs. More refined **lexicalized reordering models** estimate the parameters from the training data based on the relative distance of two phrase pairs (Tillman 2004; Galley and Manning 2008).

2.3.2 Sparse features. Although dense features form the foundation of most SMT systems, in recent years the ability to define richer feature sets and directly optimize the system using rich features has been shown to allow for significant increases in accuracy. On the other hand, large and sparse feature sets make the MT optimization problem significantly harder, and many of the optimization methods we will cover in the rest of this survey are aimed at optimizing rich feature sets.

The first variety of sparse features that we can think of are **phrase features** or **rule features**, which count the occurrence of every phrase or rule. Of course, it is only possible to learn parameters for a translation rule if it exists in the training data used in optimization, so when using a smaller data set for optimization it is difficult to robustly learn these features. Chiang, Knight, and Wang (2009) have noted that this problem can be alleviated by only selecting and optimizing the more frequent of the sparse features. Simianer, Riezler, and Dyer (2012) also propose features using the "shape" of translation rules, transforming a rule

$$\text{X} \rightarrow \langle \text{ne X}_{\boxed{1}} \text{ pas, did not X}_{\boxed{1}} \rangle \tag{5}$$

into a string simply indicating whether each word is a terminal (T) or non-terminal (N)

$$\text{N} \rightarrow \langle \text{T N T, T T N} \rangle \tag{6}$$

Count-based features can also be extended to cover other features of the translation, such as phrase or rule bigrams, indicating which phrases or rules tend to be used together (Simianer, Riezler, and Dyer 2012).

Another alternative for the creation of features that are sparse, but less sparse than features of phrases or rules, are **lexical features** (Watanabe et al. 2007). Lexical features,

similar to lexical weighting, focus on the correspondence between the individual words that are included in a phrase or rule. The simplest variety of lexical features remembers which source words f are aligned with which target words e, and fires a feature for each pair. It is also possible to condition lexical features on the surrounding context in the source language (Chiang, Knight, and Wang 2009; Xiao et al. 2011), fire features between every pair of words in the source or target sentences (Watanabe et al. 2007), or integrate bigrams on the target side (Watanabe et al. 2007). Of these, the former two can be calculated from source and local target context, but target bigrams require target bigram context and are thus non-local features.

One final variety of features that has proven useful is **syntax-based features** (Blunsom and Osborne 2008; Marton and Resnik 2008). In particular, phrase-based and hierarchical phrase-based translations do not directly consider syntax (in the linguistic sense) in the construction of the models, so introducing this information in the form of features has a potential for benefit. One way to introduce this information is to parse the input sentence before translation, and use the information in the parse tree in the calculation of features. For example, we can count the number of times a phrase or translation rule matches, or partially matches (Marton and Resnik 2008), a span with a particular label, based on the assumption that rules that match a syntactic span are more likely to be syntactically reasonable.

2.3.3 Summary features. Although sparse features are useful, training of sparse features is an extremely difficult optimization problem, and at this point there is still no method that has been widely demonstrated as being able to robustly estimate the parameters of millions of features. Because of this, a third approach of first training the parameters of sparse features, then condensing the sparse features into dense features and performing one more optimization pass (potentially with a different algorithm), has been widely used in a large number of research papers and systems (Dyer et al. 2009; He and Deng 2012; Flanigan, Dyer, and Carbonell 2013; Setiawan and Zhou 2013). A dense feature created from a large group of sparse features and their weights is generally called a **summary feature**, and can be expressed as follows

$$h_{\text{sum}}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}) = \boldsymbol{w}_{\text{sparse}}^{\top} \boldsymbol{h}_{\text{sparse}}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}) \tag{7}$$

There has also been work that splits sparse features into not one, but multiple groups, creating a dense feature for each group (Xiang and Ittycheriah 2011; Liu et al. 2013).

2.4 Decoding

Given an input sentence $\boldsymbol{f}$, the task of decoding is defined as an inference problem of finding the best scoring derivation $\langle \hat{\boldsymbol{e}}, \hat{\boldsymbol{d}} \rangle$ according to Equation (1). In general, the inference is intractable if we enumerate all possible derivations in $\mathcal{T}(\boldsymbol{f})$ and rank each derivation by the model. We assume that a derivation is composed of a set of steps

$$\boldsymbol{d} = d_1, d_2, \cdots, d_{|\boldsymbol{d}|} \tag{8}$$

where each d_j is a step—for example, a phrase pair in phrase-based MT or a synchronous rule in tree-based MT—ordered in a particular way. We also assume that

each feature function can be decomposed over each step, and Equation (1) can be expressed by

$$\langle \hat{\boldsymbol{e}}, \hat{\boldsymbol{d}} \rangle = \underset{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \mathcal{T}(\boldsymbol{f})}{\arg\max} \sum_{i}^{|\boldsymbol{w}|} w_i \sum_{j=1}^{|\boldsymbol{d}|} h_i(d_j, \rho_i(d_1^{j-1})) \tag{9}$$

where $h_i(d_j, \rho_i(d_1^{j-1}))$ is a feature function for the jth step decomposed from the global feature function of $h_i(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d})$. As mentioned in the previous section, non-local features require information that cannot be calculated directly from the rule itself, and $\rho_i(d_1^{j-1})$ is a variable that defines the residual information to score this ith feature function using the partial derivation d_1^{j-1} (Gesmundo and Henderson 2011; Green, Cer, and Manning 2014). For example, in phrase-based translation, for an n-gram language model feature, $\rho_i(d_1^{j-1})$ will be the $n-1$ word suffix of the partial translation (Koehn, Och, and Marcu 2003). The local feature functions, such as phrase translation probabilities in Section 2.3.1, require no context from partial derivations, and thus $\rho_i(d_1^{j-1}) = \emptyset$.

The problem of decoding is treated as a search problem in which partial derivations $\dot{\boldsymbol{d}}$ together with $\rho_i(\dot{\boldsymbol{d}})$ in Equation (9) are enumerated to form hypotheses or states. In phrase-based MT, search is carried out by enumerating partial derivations in left-to-right order on the target side while remembering the translated source word positions. Similarly, the search in MT with synchronous grammars is performed by using the CYK+ algorithm (Chappelier and Rajman 1998) on the source side and generating partial derivations for progressively longer source spans. Because of the enormous search space brought about by maintaining $\rho_i(\dot{\boldsymbol{d}})$ in each partial derivation, **beam search** is used to heuristically prune the search space. As a result, the search is **inexact** because of the **search error** caused by heuristic pruning, in which the best scoring hypothesis is not necessarily optimal in terms of given model parameters.

The search is efficiently carried out by merging equivalent states encoded as ρ (Koehn, Och, and Marcu 2003; Huang and Chiang 2007), and the space is succinctly represented by compact data structures, such as **graphs** (Ueffing, Och, and Ney 2002) (or **lattices**) in phrase-based MT (Koehn, Och, and Marcu 2003) and **hypergraphs** (Klein and Manning 2004) (or **packed forests**) in tree-based MT (Huang and Chiang 2007). These data structures may be directly used as compact representations of all derivations for optimization.

However, using these data structures directly can be unwieldly, and thus it is more common to obtain a k-best list as an approximation of the derivation space. Figure 1(a) shows an example of k-best English translations for a French input sentence, '*la délégation chinoise appuiera pleinement la présidence*.' The k-best list may be obtained either from a lattice in Figure 1(b) or from a forest in Figure 1(c). It should be noted that different derivations in a k-best list may share the same translation due to the variation of phrases or rules in constructing a translation, e.g., the choice of *support the chair* or *support* and *the chair* in Figure 1(b). A diverse k-best list can be obtained by extracting a unique k-best list that maintains only the best scored derivation sharing the same translation (Huang, Knight, and Joshi 2006; Hasan, Zens, and Ney 2007), by incorporating a penalty term when scoring derivations (Gimpel et al. 2013), or by performing Monte Carlo sampling to acquire a more diverse set of candidates (Blunsom and Osborne 2008).

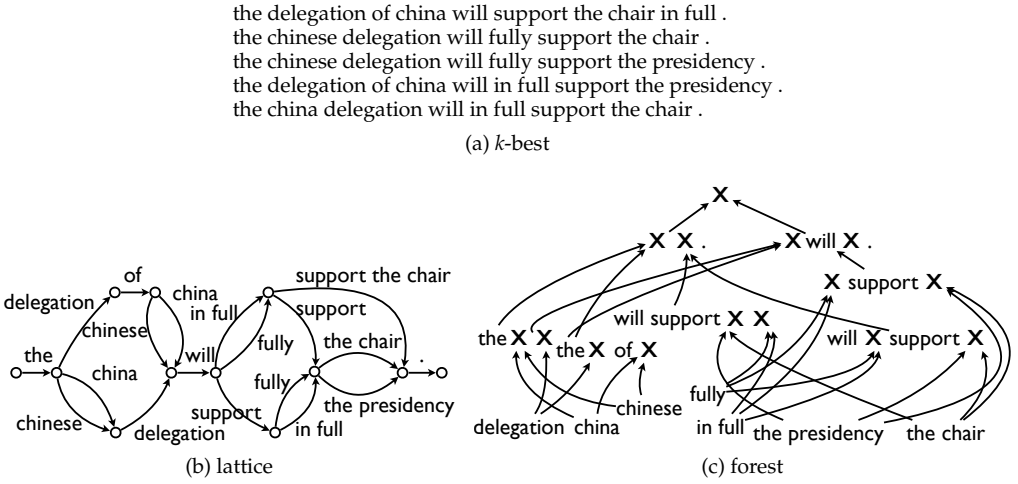


Figure 1
Example of a *k*-best list, lattice, and forest.

Another class of decoding problem is **forced decoding**, in which the output from a decoder is forced to match with a reference translation of the input sentence. In phrase-based MT, this is implemented by adding additional features to reward hypotheses that match with the given target sentence (Liang, Zhang, and Zhao 2012; Yu et al. 2013). In MT using synchronous grammars, it is carried out by **biparsing** over two languages, for instance, by a variant of the CYK algorithm (Wu 1997) or by a more efficient two-step algorithm (Dyer 2010b; Peitz et al. 2012). Even if we perform forced decoding, we are still not guaranteed that the decoder will be able to produce the reference translation (because of unknown words, reordering limits, or other factors). This problem can be resolved by preserving the prefix of partial derivations (Yu et al. 2013), or by allowing approximate matching of the target side (Liang, Zhang, and Zhao 2012). It is also possible to create a **neighborhood** of a forced decoding derivation by adding additional hyperedges to the true derivation, which allows for efficient generation of negative examples for discriminative learning algorithms (Xiao et al. 2011).

2.5 Evaluation

Once we have a machine translation system that can produce translations, we next must perform **evaluation** to judge how good the generated translations actually are. As the final consumer of machine translation output is usually a human, the most natural form of evaluation is manual evaluation by human annotators. However, because human evaluation is expensive and time-consuming, in recent years there has been a shift to automatic calculation of the quality of MT output.

In general, automatic evaluation measures use a set of data consisting of N input sentences $F = \left\{ \boldsymbol{f}^{(i)} \right\}_{i=1}^{N}$, each of which having a **reference translation** $E = \left\{ \boldsymbol{e}^{(i)} \right\}_{i=1}^{N}$ that was created by a human translator. The input F is automatically translated using a machine translation system to acquire MT results $\hat{E} = \left\{ \hat{\boldsymbol{e}}^{(i)} \right\}_{i=1}^{N}$, which are then compared to the corresponding references. The closer the MT output is to the reference, the better it is deemed to be, according to automatic evaluation. In addition, as there are often

many ways to translate a particular sentence, it is also possible to perform evaluation with multiple references created by different translators. There has also been some work on encoding a huge number of references in a lattice, created either by hand (Dreyer and Marcu 2012) or by automatic paraphrasing (Zhou, Lin, and Hovy 2006).

One major distinction between optimization measures is whether they are calculated on the **corpus level** or the **sentence level**. Corpus-level measures are calculated by taking statistics over the whole corpus, whereas sentence-level measures are calculated by measuring sentence-level accuracy, and defining the corpus-level accuracy as the average of the sentence-level accuracies. All optimization algorithms that are applicable to corpus-level measures are applicable to sentence-level measures, but the opposite is not true, making this distinction important from the optimization point of view.

The most commonly used MT evaluation measure BLEU (Papineni et al. 2002) is defined on the corpus level, and we will cover it in detail as it plays an important role in some of the methods that follow. Of course, there have been many other evaluation measures proposed since BLEU, with TER (Snover et al. 2006) and METEOR (Banerjee and Lavie 2005) being among the most widely used. The great majority of metrics other than BLEU are defined on the sentence level, and thus are conducive to optimization algorithms that require sentence-level evaluation measures. We discuss the role of evaluation in MT optimization more completely in Section 8.3.

2.5.1 BLEU. BLEU is defined as the geometric mean of n-gram precisions (usually for n from 1 to 4), and a brevity penalty to prevent short sentences from receiving unfairly high evaluation scores. For a single reference sentence $\boldsymbol{e}$ and a corresponding system output $\hat{\boldsymbol{e}}$, we can define $c_n(\hat{\boldsymbol{e}})$ as the number of n-grams in $\hat{\boldsymbol{e}}$, and $m_n(\boldsymbol{e}, \hat{\boldsymbol{e}})$ as the number of n-grams in $\hat{\boldsymbol{e}}$ that match $\boldsymbol{e}$

$$c_n(\hat{\boldsymbol{e}}) = |\{g_n \in \hat{\boldsymbol{e}}\}|$$
$$m_n(\boldsymbol{e}, \hat{\boldsymbol{e}}) = |\{g_n \in \hat{\boldsymbol{e}}\} \cap \{g'_n \in \boldsymbol{e}\}|$$

Here, $\{g_n \in \hat{\boldsymbol{e}}\}$ and $\{g'_n \in \boldsymbol{e}\}$ are multisets that can contain identical n-grams more than once, and $\cap$ is an operator for multisets that allows for consideration of multiple instances of the same n-gram.[3] Note that the total count for a candidate n-gram is **clipped** to be no more than the count in the reference translation. If we have a corpus of reference sets $R = \{\boldsymbol{e}^{(1)}, \ldots, \boldsymbol{e}^{(N)}\}$, where each sentence has M references $\boldsymbol{e}^{(i)} = \{\boldsymbol{e}_1^{(i)}, \ldots, \boldsymbol{e}_M^{(i)}\}$, the BLEU score of the corresponding system outputs $E = \{\hat{\boldsymbol{e}}^{(1)}, \ldots, \hat{\boldsymbol{e}}^{(N)}\}$ can be defined as

$$\text{BLEU}(E, \hat{E}) = \prod_{n=1}^{4} \left(\frac{\sum_{i=1}^{N} m_n(\{\boldsymbol{e}_1^{(i)}, \ldots, \boldsymbol{e}_M^{(i)}\}, \hat{\boldsymbol{e}}^{(i)})}{\sum_{i=1}^{N} c_n(\hat{\boldsymbol{e}}^{(i)})} \right)^{\frac{1}{4}} \cdot \text{BP}(E, \hat{E}) \tag{10}$$

where the first term corresponds to geometric mean of the n-gram precisions, and the second term $\text{BP}(E, \hat{E})$ is the **brevity penalty**. The brevity penalty is necessary here because evaluation of precision favors systems that output only the words and phrases that have high accuracy, and avoids outputting more difficult-to-translate content that

3 We let $\#_A(a)$ denote the number of times a appeared in a multiset A, and define: $|A| = \sum_a \#_A(a)$, $\#_{A \cup B}(a) = \max\{\#_A(a), \#_B(a)\}$, and $\#_{A \cap B}(a) = \min\{\#_A(a), \#_B(a)\}$.

might not match the reference. The brevity penalty prevents this by discounting outputs that are shorter than the reference

$$\text{BP}(E, \hat{E}) = \min\left\{1, \exp\left(1 - \frac{\sum_{i=1}^{N} |\tilde{e}^{(i)}|}{\sum_{i=1}^{N} |\hat{e}^{(i)}|}\right)\right\} \tag{11}$$

where $\tilde{e}^{(i)}$ is defined as the longest reference with a length shorter than or equal to $\hat{e}^{(i)}$.

2.5.2 BLEU+1. One thing to notice here is that BLEU is calculated by taking statistics over the entire corpus, and thus it is a corpus-level measure. There is nothing inherently preventing us from calculating BLEU on a single sentence, but in the single-sentence case it is common for the number of matches of higher order n-grams to become zero, resulting in a BLEU score of zero for the entire sentence. One common solution to this problem is the use of a smoothed version of BLEU, commonly referred to as BLEU+1 (Lin and Och 2004). In BLEU+1, we add one to the numerators and denominators of each n-gram of order greater than one

$$c'_n(\hat{e}) = |\{g_n \in \hat{e}\}| + \delta(n > 1)$$
$$m'_n(e, \hat{e}) = |\{g_n \in \hat{e}\} \cap \{g'_n \in e\}| + \delta(n > 1)$$

where $\delta(\cdot)$ is a function that takes a value of 1 when the corresponding statement is true. We can then re-define a sentence-level BLEU using these smoothed counts

$$\text{BLEU}'(e, \hat{e}) = \prod_{n=1}^{4} \left(\frac{m'_n(\{e_1, \ldots, e_M\}, \hat{e})}{c'_n(\hat{e})}\right)^{\frac{1}{4}} \cdot \text{BP}(e, \hat{e}) \tag{12}$$

and the corpus-level evaluation can be re-defined as the average of sentence level evaluations

$$\text{BLEU}'(E, \hat{E}) = \frac{1}{N} \sum_{i=1}^{N} \text{BLEU}'(e^{(i)}, \hat{e}^{(i)}) \tag{13}$$

It has also been noted, however, that the average of sentence-level BLEU+1 is not a very accurate approximation of corpus-level BLEU, but by adjusting the smoothing heuristics it is possible to achieve a more accurate approximation (Nakov, Guzman, and Vogel 2012).

2.6 The Optimization Setting

During the optimization process, we will assume that we have some data consisting of sources $F = \left\{f^{(i)}\right\}_{i=1}^{N}$ with corresponding references $E = \left\{e^{(i)}\right\}_{i=1}^{N}$ as defined in the previous section, and that we would like to use these to optimize the parameters of the model. As mentioned in Section 2.5, it is also possible to use more than one reference translation in evaluation, but in this survey we will assume for simplicity of exposition that only one reference is used.

When we select a certain weight vector w, this will affect the scores calculated according to the model, and thus the result acquired during decoding, as described

in Section 2.4. To express whether this effect is a positive or negative one, we define a **loss function** $\ell(F, E; \boldsymbol{w}) : \mathcal{F}^N \times \mathcal{E}^N \times \mathbb{R}^M \rightarrow \mathbb{R}$ that provides a numerical indicator of how "bad" the translations generated when we use a particular $\boldsymbol{w}$ are. As the goal of optimization is to achieve better translations, we would like to choose parameters that reduce this loss. More formally, we can cast the problem as minimizing the expectation of $\ell(\cdot)$, or **risk minimization**:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^M} \mathbb{E}_{Pr(F,E)}[\ell(F, E; \boldsymbol{w})] \tag{14}$$

Here, $Pr(F, E)$ is the true joint distribution over all sets of input and output sentences that we are likely to be required to translate. However, in reality we will not know the true distribution over all sets of sentences a user may ask us to translate. Instead, we have a single set of data (henceforth, **training data**), and attempt to find the $\boldsymbol{w}$ that minimizes the loss on this data:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^M} \ell(F, E; \boldsymbol{w}) \tag{15}$$

Because we are now optimizing on a single empirically derived set of training data, this framework is called **empirical risk minimization**.

In machine learning problems, it is common to introduce **regularization** to prevent the learning of parameters that over-fit the training data. This gives us the framework of **regularized empirical risk minimization**, which will encompass most of the methods described in this survey, and is formalized as

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^M} \ell(F, E; \boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \tag{16}$$

where λ is a parameter adjusting the strength of regularization, and $\Omega(\boldsymbol{w})$ is a regularization term, common choices for which include the L_2 regularizer $\Omega_2(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 = \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}$ or the L_1 regularizer $\Omega_1(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_{m=1}^{M} |w_m|$ (Tibshirani 1996; Chen and Rosenfeld 1999). Intuitively, if λ is set to a small value, optimization will attempt to learn a $\boldsymbol{w}$ that effectively minimizes loss on the training data, but there is a risk of over-fitting reducing generalization capability. On the other hand, if λ is set to a larger value, optimization will be less aggressive in minimizing loss on the training data, reducing over-fitting, but also possibly failing to capture useful information that could be used to improve accuracy.

3. Defining a Loss Function

The first step in performing optimization is defining the loss function that we are interested in optimizing. The choice of a proper loss function is critical in that it effects the final performance of the optimized MT system, and also the possible choices for optimization algorithms. This section describes several common choices for loss functions, and describes their various features.

3.1 Error

The first, and most straightforward, loss that we can attempt to optimize is **error** (Och 2003). We assume that by comparing the decoder's translation result $\hat{E}$ with the reference E, we are able to calculate a function $\text{error}(E, \hat{E}) : \mathcal{E}^N \times \mathcal{E}^N \rightarrow \mathbb{R}_{\geq 0}$ that describes the extent of error included in the translations. For example, if we use the BLEU described in Section 2.5 as an evaluation measure for our system, it is natural to use $1 - \text{BLEU}$ as an error function, so that as our evaluation improves, the error decreases. Converting this to a loss function that is dependent on the model parameters, we obtain the following loss expressing the error over the 1-best results obtained by decoding in Equation (1):

$$\ell_{\text{error}}(F, E, C; \boldsymbol{w}) = \text{error}\left(E, \left\{ \underset{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{c}^{(i)}}{\arg\max}\, \boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) \right\}_{i=1}^{N}\right) \tag{17}$$

Error has the advantage of being simple, easy to explain, and directly related to translation performance, and these features make it perhaps the most commonly used loss in current machine translation systems. On the other hand, it also has a large disadvantage in that the loss function expressed in Equation (17) is not convex, and most MT evaluation measures used in the calculation of the error function $\text{error}(\cdot)$ are not continuously differentiable. This makes direct minimization of error a difficult optimization problem (particularly for larger feature sets), and thus a number of other, easier-to-optimize losses are used as well.

A special instance of error, which is worth mentioning because of its relation to the methods we will introduce in the following sections, is **zero–one loss**. Zero–one loss focuses on whether an **oracle translation** is chosen as the system output. Oracle translations can be vaguely defined as "good" translations, such as the reference translation $\boldsymbol{e}^{(i)}$, or perhaps the best translation in the k-best list (described in detail in Section 4). If we define the set of oracle translations for sentence i as $\boldsymbol{o}^{(i)}$, zero–one loss is defined by plugging the following zero–one error function into Equation (17):

$$\text{error}(E, \hat{E}) = \frac{1}{N} \sum_{i=1}^{N} \left(1 - \delta(\hat{\boldsymbol{e}}^{(i)} \in \boldsymbol{o}^{(i)})\right) \tag{18}$$

where $\hat{\boldsymbol{e}}^{(i)}$ is the one-best translation candidate, and $\delta(\hat{\boldsymbol{e}}^{(i)} \in \boldsymbol{o}^{(i)})$ is one if $\hat{\boldsymbol{e}}^{(i)}$ is a member of $\boldsymbol{o}^{(i)}$ and zero otherwise.

3.2 Softmax Loss

One thing to note about error is that there is no concept of "probability" of each translation candidate incorporated in its calculation. Being able to define a well-scaled probability of candidates can be useful, however, for estimation of confidence measures or incorporation with downstream applications. **Softmax loss** is a loss that is similar to the zero–one loss, but directly defines a probabilistic model and attempts to maximize the probability of the oracle translations (Berger, Della Pietra, and Della Pietra 1996; Och and Ney 2002; Blunsom, Cohn, and Osborne 2008).

In particular, if we assume that MT is modeled according to the log-linear model

$$p_{\boldsymbol{w}}(\boldsymbol{e}, \boldsymbol{d}|\boldsymbol{f}, \boldsymbol{c}) = \frac{\exp(\boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}))}{\sum_{\langle \boldsymbol{e}', \boldsymbol{d}' \rangle \in \boldsymbol{c}} \exp(\boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}', \boldsymbol{d}'))} \tag{19}$$

we can define softmax loss $\ell_{\text{softmax}}(\cdot)$ as follows:

$$\ell_{\text{softmax}}(F, E, C; \boldsymbol{w}) = -\frac{1}{N} \prod_{i=1}^{N} \sum_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{o}^{(i)}} p_{\boldsymbol{w}}(\boldsymbol{e}, \boldsymbol{d}|\boldsymbol{f}^{(i)}, \boldsymbol{c}^{(i)}) \tag{20}$$

$$= -\frac{1}{N} \prod_{i=1}^{N} \frac{\sum_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{o}^{(i)}} \exp(\boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}))}{\sum_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{c}^{(i)}} \exp(\boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}))} \tag{21}$$

From Equation (21) we can see that only the oracle translations contribute to the numerator, and all candidates in $\boldsymbol{c}^{(i)}$ contributes to the denominator. Thus, intuitively, the softmax objective prefers parameter settings that assign high scores to the oracle translations, and lower scores to any other members of $\boldsymbol{c}^{(i)}$ that are not oracles.

It should be noted that this loss can be calculated from a k-best list by iterating over the entire list and calculating the numerators and denominators in Equation (19). It is also possible, but more involved, to calculate over lattices or forests by using dynamic programming algorithms such as the forward–backward or inside–outside algorithms (Blunsom, Cohn, and Osborne 2008; Gimpel and Smith 2009).

3.3 Risk-Based Loss

In contrast to softmax loss, which can be viewed as a probabilistic version of zero–one loss, **risk** defines a probabilistic version of the translation error (Smith and Eisner 2006; Zens, Hasan, and Ney 2007; Li and Eisner 2009; He and Deng 2012). Specifically, risk is based on the expected error incurred by a probabilistic model parameterized by $\boldsymbol{w}$. This combines the advantages of the probabilistic model in softmax loss with the direct consideration of translation accuracy afforded by using error directly. In comparison to error, it also has the advantage of being differentiable, allowing for easier optimization.

To define this error, we define a scaling parameter $\gamma \geq 0$ and use it in the calculation of each hypothesis's probability

$$p_{\gamma, \boldsymbol{w}}(\boldsymbol{e}, \boldsymbol{d}|\boldsymbol{f}, \boldsymbol{c}) = \frac{\exp(\gamma \boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d}))}{\sum_{\langle \boldsymbol{e}', \boldsymbol{d}' \rangle \in \boldsymbol{c}} \exp(\gamma \boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}', \boldsymbol{d}'))} \tag{22}$$

Given this probability, we then calculate the expected loss as follows:

$$\ell_{\text{risk}}(F, E, C; \gamma, \boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{p_{\gamma, \boldsymbol{w}}(\boldsymbol{e}, \boldsymbol{d}|\boldsymbol{f}^{(i)}, \boldsymbol{c}^{(i)})}[\text{err}(\boldsymbol{e}^{(i)}, \boldsymbol{e})] \tag{23}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{c}^{(i)}} \text{err}(\boldsymbol{e}^{(i)}, \boldsymbol{e}) p_{\gamma, \boldsymbol{w}}(\boldsymbol{e}, \boldsymbol{d}|\boldsymbol{f}^{(i)}, \boldsymbol{c}^{(i)}) \tag{24}$$

In Equation (22), when $\gamma = 0$ regardless of parameters $\boldsymbol{w}$ every hypothesis $\langle \boldsymbol{e}, \boldsymbol{d} \rangle$ will be assigned a uniform probability, and when $\gamma = 1$ the probabilities are equivalent to those in the log-linear model of Equation (19). When $\gamma \to \infty$, the probability of the highest-scored hypothesis will approach 1, and thus our objective will approach the error defined in Equation (17). This γ can be adjusted in a way that allows for more effective search of the parameter space, as described in more detail in Section 5.5.

3.4 Margin-Based Loss

The zero–one loss in Section 3.1 was based on whether the oracle received a higher score than other hypotheses. The idea of **margin**, which is behind the classification paradigm of **support vector machines** (SVMs) (Joachims 1998), takes this a step further, finding parameters that explicitly maximize the distance, or margin, between correct and incorrect candidates. The main advantage of margin-based methods is that they are able to consider the error function, and often achieve high accuracy. These advantages make margin-based methods perhaps the second most popular loss used in current MT systems after direct minimization of error.

This margin-based objective can be defined as the loss:

$$\ell_{\text{margin}}(F, E, C; \boldsymbol{w}) = \frac{1}{\mathcal{N}(C)} \sum_{i=1}^{N} \sum_{\langle \boldsymbol{e}^*, \boldsymbol{d}^* \rangle \in \boldsymbol{o}^{(i)}} \sum_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{c}^{(i)} \setminus \boldsymbol{o}^{(i)}} \max \left\{ 0, \Delta \operatorname{err}(\boldsymbol{e}^{(i)}, \boldsymbol{e}^*, \boldsymbol{e}) - \boldsymbol{w}^\top \Delta \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}^*, \boldsymbol{d}^*, \boldsymbol{e}, \boldsymbol{d}) \right\} \quad (25)$$

where we define

$$\Delta \operatorname{err}(\boldsymbol{e}, \boldsymbol{e}^*, \boldsymbol{e}') = \operatorname{err}(\boldsymbol{e}, \boldsymbol{e}') - \operatorname{err}(\boldsymbol{e}, \boldsymbol{e}^*) \quad (26)$$

$$\Delta \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}^*, \boldsymbol{d}^*, \boldsymbol{e}', \boldsymbol{d}') = \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}^*, \boldsymbol{d}^*) - \boldsymbol{h}(\boldsymbol{f}, \boldsymbol{e}', \boldsymbol{d}') \quad (27)$$

In Equation (25), we first specify that for each pair of oracle candidates $\boldsymbol{o}^{(i)}$, and non-oracle candidates $\boldsymbol{c}^{(i)} \setminus \boldsymbol{o}^{(i)}$, the margin $\boldsymbol{w}^\top \Delta \boldsymbol{h}(\cdot)$ between oracle $\boldsymbol{e}^*$ and non-oracle $\boldsymbol{e}$ should be greater than the difference in the error $\Delta \operatorname{err}(\cdot)$.[4] We then define the loss as the total amount that this margin is violated. In this loss calculation, the number of pairs is $\mathcal{N}(C) = \sum_{i=1}^{N} |\boldsymbol{c}^{(i)} \setminus \boldsymbol{o}^{(i)}| \cdot |\boldsymbol{o}^{(i)}|$. Note that here $\operatorname{err}(\cdot)$ is not calculated on the corpus level, but on the sentence level, and may not directly correspond to our corpus-level error $\operatorname{error}(\cdot)$.

It is also common to consider the case where we calculate this loss with regards to only a single translation candidate and oracle, and this is often called **hinge loss**. If we define $\langle \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)} \rangle \in \boldsymbol{c}^{(i)} \setminus \boldsymbol{o}^{(i)}$ as the 1-best translation candidate

$$\langle \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)} \rangle = \operatorname*{arg\,max}_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{c}^{(i)} \setminus \boldsymbol{o}^{(i)}} \boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) \quad (28)$$

4 Equation (25) can be regarded as an instance of ranking loss described in Section 3.5 in which better translations are selected only from a set of oracle translations.

and $\langle \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)} \rangle \in \boldsymbol{o}^{(i)}$ as the oracle translation

$$\langle \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)} \rangle = \operatorname*{arg\,min}_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{o}^{(i)}} \operatorname{err}(\boldsymbol{e}^{(i)}, \boldsymbol{e}) \tag{29}$$

the hinge loss can be defined as follows

$$\ell_{\text{hinge}}(F, E, C; \boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \max \left\{ 0, \Delta \operatorname{err}(\boldsymbol{e}^{*(i)}, \boldsymbol{e}^{*}, \boldsymbol{e}) - \boldsymbol{w}^{\top} \Delta \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)}, \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)}) \right\} \tag{30}$$

A special instance of this hinge loss that is widely used in machine translation, and machine learning in general, is **perceptron loss** (Liang et al. 2006), which further removes the term considering the error, and simply incurs a penalty if the 1-best candidate receives a higher score than the oracle

$$\ell_{\text{perceptron}}(F, E, C; \boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \max \left\{ 0, -\boldsymbol{w}^{\top} \Delta \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)}, \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)}) \right\} \tag{31}$$

In addition to maximizing the margin itself, there has also been work on maximizing the **relative margin** (Eidelman, Marton, and Resnik 2013). To explain the relative margin, we first define the **worst hypothesis** as

$$\langle \check{\boldsymbol{e}}^{(i)}, \check{\boldsymbol{d}}^{(i)} \rangle = \operatorname*{arg\,max}_{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \in \boldsymbol{c}^{(i)}} \operatorname{err}(\boldsymbol{e}^{(i)}, \boldsymbol{e}) \tag{32}$$

and then calculate the **spread** $\Delta \operatorname{err}(\boldsymbol{e}^{(i)}, \dot{\boldsymbol{e}}^{(i)}, \check{\boldsymbol{e}}^{(i)})$, which is the difference of errors between the oracle hypothesis $\langle \dot{\boldsymbol{e}}^{(i)}, \dot{\boldsymbol{d}}^{(i)} \rangle$ and worst hypothesis $\langle \check{\boldsymbol{e}}^{(i)}, \check{\boldsymbol{d}}^{(i)} \rangle$. An additional term can then be added to the objective function to penalize parameter settings with large spreads. The intuition behind the relative margin criterion is that in addition to increasing the margin, considering the spread reduces the variance between the non-oracle hypotheses. Given an identical margin, having a smaller variance indicates that an unseen hypothesis will be less likely to pass over the margin and be misclassified.

3.5 Ranking Loss

Perceptron and margin losses attempted to distinguish between oracle and non-oracle hypotheses. This can be considered a specific instance of the more general **ranking** framework (Herbrich, Graepel, and Obermayer 1999; Freund et al. 2003; Burges et al. 2005; Cao et al. 2007), where, for an arbitrary pair of translation candidates, a binary classifier is trained to distinguish which of the two candidates has the lower error. If a

particular pair of candidates in the training data $\langle e_k, d_k \rangle$ and $\langle e_{k'}, d_{k'} \rangle$ is ranked in the correct order, the following condition is satisfied:

$$\begin{aligned} &\text{err}(e^{(i)}, e_k) < \text{err}(e^{(i)}, e_{k'}) \\ &\iff w^\top h(f^{(i)}, e_k, d_k) > w^\top h(f^{(i)}, e_{k'}, d_{k'}) \end{aligned}$$

This can be expressed as

$$\begin{aligned} &\text{err}(e^{(i)}, e_k) < \text{err}(e^{(i)}, e_{k'}) \\ &\iff w^\top h(f^{(i)}, e_k, d_k) > w^\top h(f^{(i)}, e_{k'}, d_{k'}) \\ &\iff w^\top h(f^{(i)}, e_k, d_k) - w^\top h(f^{(i)}, e_{k'}, d_{k'}) > 0 \\ &\iff w^\top \Big(h(f^{(i)}, e_k, d_k) - h(f^{(i)}, e_{k'}, d_{k'}) \Big) > 0 \\ &\iff w^\top \Delta h(f^{(i)}, e_k, d_k, e_{k'}, d_{k'}) > 0 \end{aligned}$$

where $\Delta h(f^{(i)}, e_k, d_k, e_{k'}, d_{k'})$ can be treated as training data to be classified using any variety of **binary classifier**. Each binary decision made by this classifier becomes an individual choice, and thus the ranking loss is the sum of these individual losses. As the binary classifier, it is possible to use perceptron, hinge, or softmax losses between the correct and incorrect answers.

It should be noted that standard ranking techniques make a hard decision between candidates with higher and lower error, which can cause problems when the ranking by error does not correlate well with the ranking measured by the model. The **cross-entropy ranking loss** solves this problem by softly fitting the model distribution to the distribution of ranking measured by errors (Green et al. 2014).

3.6 Mean Squared Error Loss

Finally, **mean squared error loss** is another method that does not make a hard zero–one decision between the better and worse candidates, but instead attempts to directly estimate the difference in scores (Bazrafshan, Chung, and Gildea 2012). This is done by first finding the difference in errors between the two candidates $\Delta \text{err}(e^{(i)}, e^*, e)$ and defining the loss as the mean squared error of the difference between the inverse of the difference in the errors and the difference in the model scores[5]:

$$\ell_{\text{mse}}(F, E, C; w) = \frac{1}{\mathcal{N}(C)} \sum_{i=1}^{N} \sum_{\langle e^*, d^* \rangle \in o^{(i)}} \sum_{\langle e, d \rangle \in c^{(i)}} \Big(-\Delta \text{err}(e^{(i)}, e^*, e) - w^\top \Delta h(f^{(i)}, e^*, d^*, e, d) \Big)^2 \quad (33)$$

5 We take the inverse because we would like model scores and errors to be inversely correlated.

4. Choosing Oracles

In the previous section, many loss functions used **oracle translations**, which are defined as a set of translations for any sentence that are "good." Choosing oracle translations is not a trivial task, and in this section we describe the details involved.

4.1 Bold vs. Local Updates

In other structured learning tasks such as part-of-speech tagging or parsing, it is common to simply use the correct answer as an oracle. In translation, this is equivalent to optimizing towards an actual human reference, which is called **bold update** (Liang et al. 2006). It should be noted that even if we know the reference $\boldsymbol{e}$, we still need to obtain a derivation $\boldsymbol{d}$, and thus it is necessary to perform forced decoding (described in Section 2.4) to obtain this derivation.

However, bold update has a number of practical difficulties. For example, we are not guaranteed that the decoder is able to actually produce the reference (for example, in the case of unknown words), in which case forced decoding will fail. In addition, even if the hypothesis exists in the search space, it might require a large change in parameters $\boldsymbol{w}$ to ensure that the reference gets a higher score than all other hypotheses. This is true in the case of non-literal translations, for example, which may be producible by the decoder, but only by using a derivation that would normally receive an extremely low probability.

Local update is an alternative method that selects an oracle from a set of hypotheses produced during the normal decoding process. The space of hypotheses used to select oracles is usually based on k-best lists, but can also include lattices or forests output by the decoder as described in Section 2.4. Because of the previously mentioned difficulties with bold update, it has been empirically observed that local update tends to outperform bold update in online optimization (Liang et al. 2006). However, it also makes it necessary to select oracle translations from a set of imperfect decoder outputs, and we will describe this process in more detail in the following section.

4.2 Selecting Oracles and Approximating Corpus-Level Errors

First, we define $\boldsymbol{o}^{(i)} \subseteq \boldsymbol{c}^{(i)}$ as the set of oracle translations, derivation-translation pairs in $\boldsymbol{c}^{(i)}$ that minimize the error function

$$O = \underset{\left\{\boldsymbol{o}^{(i)} \subseteq \boldsymbol{c}^{(i)}\right\}_{i=1}^{N}}{\arg\min} \ \text{error}\left(E, \left\{\boldsymbol{o}^{(i)} \subseteq \boldsymbol{c}^{(i)}\right\}_{i=1}^{N}\right) \tag{34}$$

One thing to note here is that error$(\cdot)$ is a corpus-level error function. As mentioned in Section 2.5, evaluation measures for MT can be classified into those that are decomposable on the sentence level, and those that are not. If this error function can be composed as the sum of sentence-level errors, such as BLEU+1, choosing the oracle is simple; we simply need to find the set of candidates that have the lowest error independently sentence by sentence.[6]

6 More accurately, finding the oracle in the k-best list by enumeration of the hypotheses is easy, but finding the oracle in a compressed data structure such as a lattice is computationally difficult, and approximation algorithms are necessary (Leusch, Matusov, and Ney 2008; Li and Khudanpur 2009; Sokolov, Wisniewski, and Yvon 2012a).

1: **procedure** ORACLE($\langle F, E, C \rangle$)
2: $O = \{\boldsymbol{o}^{(i)}\}_{i=1}^{N}$
3: $\boldsymbol{o}^{(i)} \leftarrow \{\langle \boldsymbol{e}, \boldsymbol{d} \rangle \sim \boldsymbol{c}^{(i)}\}$ or $\boldsymbol{o}^{(i)} \leftarrow \emptyset$
4: **repeat**
5: **for** $i \in$ PERMUTE($\{1, \ldots, N\}$) **do** ▷ Random order
6: $\boldsymbol{o}^{(i)} \leftarrow \emptyset$
7: $s \leftarrow \infty$
8: **for** $k \in \{1, \ldots, K\}$ **do**
9: $s' \leftarrow \text{error}\left(E, \{\boldsymbol{o}_1^{(1)}, \ldots, \boldsymbol{o}_1^{(i-1)}, \boldsymbol{c}_k^{(i)}, \boldsymbol{o}_1^{(i+1)}, \ldots, \boldsymbol{o}_1^{(N)}\}\right)$
10: **if** $s' < s$ **then** ▷ Update the oracle
11: $\boldsymbol{o}^{(i)} \leftarrow \{\boldsymbol{c}_k^{(i)}\}$
12: $s \leftarrow s'$
13: **else if** $s' = s$ **then** ▷ Same error value
14: $\boldsymbol{o}^{(i)} \leftarrow \boldsymbol{o}^{(i)} \cup \{\boldsymbol{c}_k^{(i)}\}$
15: **end if**
16: **end for**
17: **end for**
18: **until convergence** ▷ If O doesn't change, converged
19: **return** O
20: **end procedure**

Figure 2
Greedy search for an oracle.

However, when using a corpus-level error function we need a slightly more sophisticated method, such as the **greedy method** of Venugopal and Vogel (2005). In this method (Figure 2), the oracle is first initialized either as an empty set or by randomly picking from the candidates. Next, we iterate randomly through the translation candidates in $\boldsymbol{c}^{(i)}$, try replacing the current oracle $\boldsymbol{o}^{(i)}$ with the candidate, and check the change in the error function (Line 9), and if the error decreases, replace the oracle with the tested candidate. This process is repeated until there is no change in O.

4.3 Selecting Oracles for Margin-Based Methods

Considering the hinge loss of Equation (30), the 1-best and oracle candidates are acquired according to Equation (28) and Equation (29), respectively, and the loss is calculated according to Equation (31). In order to minimize this loss, we would like to select the pair with the largest loss, and update so that the loss gets smaller. However, it is not necessarily the case that the candidates with the maximum model score $\langle \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)} \rangle$ and minimum loss $\langle \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)} \rangle$ form the pair with the minimal margin. Thus, when using margin-based objectives, it is common to modify the criterion for selecting candidates

to use in the update as follows (Chiang, Marton, and Resnik 2008; Chiang, Knight, and Wang 2009):

$$\langle \bar{e}^{(i)}, \bar{d}^{(i)} \rangle = \arg\max_{\langle e,d \rangle \in c^{(i)}} w^\top h(f^{(i)}, e, d) + \text{err}(e^{(i)}, e) \tag{35}$$

$$\langle \dot{e}^{(i)}, \dot{d}^{(i)} \rangle = \arg\max_{\langle e,d \rangle \in c^{(i)}} w^\top h(f^{(i)}, e, d) - \text{err}(e^{(i)}, e) \tag{36}$$

Thus, we can replace $\langle \hat{e}^{(i)}, \hat{d}^{(i)} \rangle$ and $\langle e^{*(i)}, d^{*(i)} \rangle$ with $\langle \bar{e}^{(i)}, \bar{d}^{(i)} \rangle$ and $\langle \dot{e}^{(i)}, \dot{d}^{(i)} \rangle$, resulting in a margin of

$$\Delta \text{err}(e^{(i)}, \dot{e}^{(i)}, \bar{e}^{(i)}) - w^\top \Delta h(f^{(i)}, \dot{e}^{(i)}, \dot{d}^{(i)}, \bar{e}^{(i)}, \bar{d}^{(i)}) \tag{37}$$

which is the largest margin in the k-best list. Explaining more intuitively, this criterion provides a bias towards selecting hypotheses with high error, making the learning algorithm work harder to correctly classify very bad hypotheses than it does for hypotheses that are only slightly worse than the oracle. Inference methods that consider the loss as in Equations (35) and (36) are called **loss-augmented inference** (Taskar et al. 2005) methods, and can minimize losses with respect to the candidate with the largest violation. Gimpel and Smith (2012) take this a step further, defining a **structured ramp loss** that additionally considers Equations (28) and (29) within this framework.

5. Batch Methods

Now that we have explained the details of calculating loss functions used in machine translation, we turn to the actual algorithms used in optimizing using these loss functions. In this section, we cover **batch learning** approaches to MT optimization. Batch learning works by considering the entire training data on every update of the parameters, in contrast to online learning (covered in the following section), which considers only part of the data at any one time. In standard approaches to batch learning, for every training example $\langle f^{(i)}, e^{(i)} \rangle$ we enumerate every translation and derivation in the respective sets $\mathcal{E}(f^{(i)})$ and $\mathcal{D}(f^{(i)})$, and attempt to adjust the parameters so we can achieve the translations with the lowest error for the entire data.

However, as mentioned previously, the entire space of derivations is too large to handle in practice. To resolve this problem, most batch learning algorithms for MT follow the general procedure shown in Figure 3, performing iterations that alternate between decoding and optimization (Och and Ney 2002). In line 6, $\text{GEN}(f^{(i)}, w^{(t)}) = \left\{ \langle e_k^{(i)}, d_k^{(i)} \rangle \right\}_{k=1}^{K}$ indicates that we use the current parameters $w^{(t)}$ to perform decoding of sentence $f^{(i)}$, and obtain a subset of all derivations. For convenience, we will assume that this subset is expressed using a k-best list $kbest^{(i)}$, but it is also possible to use lattices or forests, as explained in Section 2.4.

A k-best list with scores for each hypothesis can be used as an approximation for the distribution over potential translations of $f^{(i)}$ according to the parameters w. However, because the size of the k-best list is limited, and the presence of search

1: **procedure** BATCHLEARN($\langle F, E\rangle = \left\{\langle \boldsymbol{f}^{(i)}, \boldsymbol{e}^{(i)}\rangle\right\}_{i=1}^{N}$)
2: $\quad \boldsymbol{w}^{(1)} \leftarrow \emptyset$
3: $\quad C = \left\{\boldsymbol{c}^{(i)} \equiv \emptyset\right\}_{i=1}^{N}$ ▷ k-best List
4: $\quad$ **for** $t \in \{1 \dots T\}$ **do**
5: $\quad\quad$ **for** $i \in \{1 \dots N\}$ **do**
6: $\quad\quad\quad kbest^{(i)} \leftarrow \text{GEN}(\boldsymbol{f}^{(i)}, \boldsymbol{w}^{(t)})$ ▷ Decode with $\boldsymbol{w}^{(t)}$
7: $\quad\quad\quad \boldsymbol{c}^{(i)} \leftarrow \boldsymbol{c}^{(i)} \cup kbest^{(i)}$ ▷ k-best Merging
8: $\quad\quad$ **end for**
9: $\quad\quad \boldsymbol{w}^{(t+1)} \leftarrow \arg\min_{\boldsymbol{w}\in\mathbb{R}^M} \ell(F, E, C; \boldsymbol{w}) + \lambda\Omega(\boldsymbol{w})$ ▷ Optimization
10: $\quad$ **end for**
11: $\quad$ **return** $\boldsymbol{w}^{(T+1)}$
12: **end procedure**

Figure 3
Batch optimization.

errors in decoding means that we are not even guaranteed to find the highest-scoring hypotheses, this approximation is far from perfect. The effect of this approximation is particularly obvious if the lack of coverage of the k-best list is systematic. For example, if the hypotheses in the k-best list are all much too short, optimization may attempt to fix this by adjusting the parameters to heavily favor very long hypotheses, far overshooting the actual optimal parameters.[7]

As a way to alleviate the problems caused by this approximation, in line 7 we merge the k-best lists from multiple decoding iterations, finding a larger and more accurate set C of derivations. Given C and the training data $\langle F, E\rangle$, we perform minimization of the $\Omega(\boldsymbol{w})$ regularized loss function $\ell(\cdot)$ and obtain new parameters $\boldsymbol{w}^{(t+1)}$ (line 9). Generation of k-best lists and optimization is performed until a hard limit of T iterations is reached, or until training has converged. In this setting, usually convergence is defined as any iteration in which the merged k-best list does not change, or when the parameters $\boldsymbol{w}$ do not change (Och 2003).

Within this batch optimization framework, the most critical challenge is to find an effective way to solve the optimization problem in line 9 of Figure 3. Section 5.1 describes methods for directly optimizing the error function. There are also methods for optimizing other losses such as those based on probabilistic models (Section 5.2), error margins (Section 5.3), ranking (Section 5.4), and risk (Section 5.5).

5.1 Error Minimization

5.1.1 Minimum Error Rate Training Overview. Minimum error rate training (MERT) (Och 2003) is one of the first, and is currently the most widely used, method for MT optimization, and focuses mainly on direct minimization of the error described in Section 3.1. Because error is not continuously differentiable, MERT uses optimization methods that do not require the calculation of a gradient, such as iterative line search

7 Liu et al. (2012) propose a method to avoid over-aggressive moves in parameter space by considering the balance between increase in the evaluation score and the similarity with the parameters on the previous iteration.

1: **procedure** MERT(F, E, C)
2: $\hat{\boldsymbol{w}} \leftarrow \emptyset$
3: **for** $r \in \{1 \dots R\}$ **do**
4: $\boldsymbol{w}^{(1)} \sim \mathbb{R}^M$ ▷ Initialize randomly
5: **for** $t \in \{1 \dots T\}$ **do** ▷ Until convergence
6: **for** $m \in \{1 \dots M\}$ **do** ▷ For each dimension
7: $\hat{\gamma}^m \leftarrow \arg\min_\gamma \ell_{\text{error}}(F, E, C; \boldsymbol{w}^{(t)} + \gamma \boldsymbol{b}^m)$ ▷ Search
8: **end for**
9: $\hat{\gamma} \leftarrow \arg\min_{\hat{\gamma}^m} \ell_{\text{error}}(F, E, C; \boldsymbol{w}^{(t)} + \hat{\gamma}^m \boldsymbol{b}^m)$ ▷ Descent
10: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \hat{\gamma} \boldsymbol{b}^m$ ▷ Update
11: **end for**
12: **if** $\ell_{\text{error}}(F, E, C; \boldsymbol{w}^{(T+1)}) < \ell_{\text{error}}(F, E, C; \hat{\boldsymbol{w}})$ **then**
13: $\hat{\boldsymbol{w}} \leftarrow \boldsymbol{w}^{(T+1)}$
14: **end if**
15: **end for**
16: **return** $\hat{\boldsymbol{w}}$
17: **end procedure**

Figure 4
Minimum error rate training (MERT).

inspired by **Powell's method** (Och 2003; Press et al. 2007), or the **Downhill-Simplex method** (**Nelder-Mead method**) (Press et al. 2007; Zens, Hasan, and Ney 2007; Zhao and Chen 2009).

The algorithm for MERT using line search is shown in Figure 4. Here, we assume that $\boldsymbol{w}$ and $\boldsymbol{h}(\cdot)$ are M-dimensional, and $\boldsymbol{b}^m$ is an M-dimensional vector where the m-th element is 1 and the rest of the elements are zero. For the T iterations, we decide the dimension m of the feature vector (line 6), and for each possible weight vector $\boldsymbol{w}^{(j)} + \gamma \boldsymbol{b}^m$ choose the $\gamma \in \mathbb{R}$ that minimizes $\ell_{\text{error}}(\cdot)$ using **line search** (line 7). Then, among the γ for each of the M search dimensions, we perform an update using $\hat{\gamma}$ that affords the largest reduction in error (lines 9 and 10). This algorithm can be deemed a variety of **steepest descent**, which is a standard method used in most implementations of MERT (Koehn et al. 2007). Another alternative is a variant of **coordinate descent** (e.g., Powell's method), in which search and update is performed in each dimension.

One feature of MERT is that it is known to easily fall into local optima of the error function. Because of this, it is standard to choose R starting points (line 4), perform optimization starting at each of these starting points, and finally choose the $\hat{\boldsymbol{w}}$ that minimizes the loss from the weights acquired from each of the R random restarts. The R starting points are generally chosen so that one of the points is the best $\boldsymbol{w}$ from the previous iteration, and the remaining $R - 1$ have each element of $\boldsymbol{w}$ chosen randomly and uniformly from some interval, although it has also been shown that more intelligent choice of initial points can result in better final scores (Moore and Quirk 2008).

5.1.2 Line Search for MERT. Although the majority of this process is relatively straightforward, the line search in Line 7 of Figure 4 requires a bit more explanation. In this step, we would like to choose the γ that results in the ordering of hypotheses in $\boldsymbol{c}^{(i)}$ that achieves the lowest error. In order to do so, MERT uses an algorithm that allows for

exact enumeration of which of the K candidates in $\boldsymbol{c}^{(i)}$ will be chosen for each value of γ. Concretely, we define

$$\arg\max_{\langle \boldsymbol{e},\boldsymbol{d}\rangle \in \boldsymbol{c}^{(i)}} \left\{ \boldsymbol{w}^{(j)} + \gamma \boldsymbol{b}^{m} \right\}^{\top} \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) \tag{38}$$

$$= \arg\max_{\langle \boldsymbol{e},\boldsymbol{d}\rangle \in \boldsymbol{c}^{(i)}} \underbrace{\boldsymbol{w}^{(j)\top} \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d})}_{intercept} + \gamma \cdot \underbrace{h_m(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d})}_{slope} \tag{39}$$

$$= \arg\max_{\langle \boldsymbol{e},\boldsymbol{d}\rangle \in \boldsymbol{c}^{(i)}} a(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) + \gamma \cdot b(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) \tag{40}$$

where each hypothesis $\langle \boldsymbol{e}, \boldsymbol{d} \rangle$ in $\boldsymbol{c}^{(i)}$ of Equation (40) is expressed as a line with intercept $a(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) (= \boldsymbol{w}^{(j)\top} \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}))$ and slope $b(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) (= h_m(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}))$ with γ as a parameter. Equation (40) is a function that returns the translation candidate with the highest score. We can define a function $g(\gamma; \boldsymbol{f}^{(i)})$ that corresponds to the score of this highest-scoring candidate as follows:

$$g(\gamma; \boldsymbol{f}^{(i)}) = \max_{\langle \boldsymbol{e},\boldsymbol{d}\rangle \in \boldsymbol{c}^{(i)}} a(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) + \gamma \cdot b(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d}) \tag{41}$$

We can see that Equation (41) is a **piecewise linear** function (Papineni 1999; Och 2003), as at any given $\gamma \in \mathbb{R}$ the translation candidate with the highest score $a(\cdot) + \gamma \cdot b(\cdot)$ will be selected, and this score corresponds to the line that is in the highest position at that particular γ. In Figure 5, we show an example with the following four translation candidates:

$$\begin{array}{ll} \boldsymbol{c}_1^{(i)} : 2.5 + \gamma \cdot (-0.8), & \boldsymbol{c}_2^{(i)} : 1 + \gamma \cdot (-0.2) \\ \boldsymbol{c}_3^{(i)} : 2 + \gamma \cdot (-0.5), & \boldsymbol{c}_4^{(i)} : -0.5 + \gamma \cdot 0.2 \end{array} \tag{42}$$

If we set γ to a very small value such as $-\infty$, the candidate with the smallest slope, in this example $\boldsymbol{c}_1^{(i)}$, will be chosen. Furthermore, if we make γ gradually larger, we will

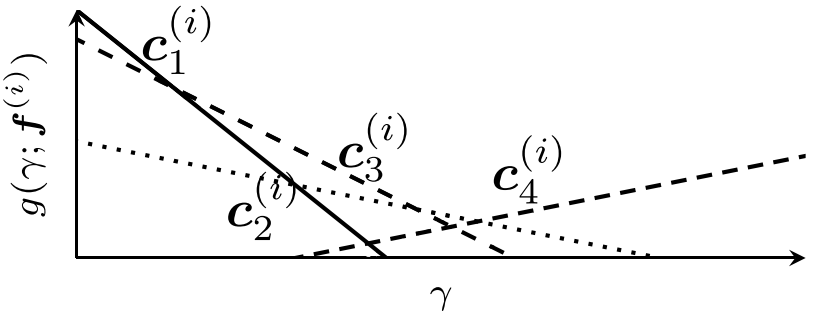


Figure 5
An example of hypotheses as lines.

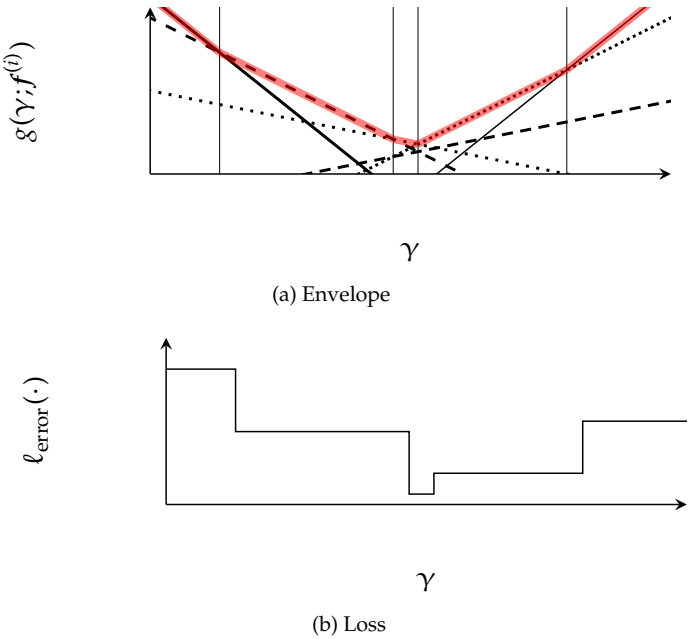
see that $\boldsymbol{c}_1^{(i)}$ continues to be the highest scoring candidate until we reach the intersection of $\boldsymbol{c}_1^{(i)}$ and $\boldsymbol{c}_3^{(i)}$ at

$$\frac{2.5 - 2}{(-0.5) - (-0.8)} \approx 1.667 \tag{43}$$

after which $\boldsymbol{c}_3^{(i)}$ will be the highest scoring candidate. If we continue increasing γ, we will continue by selecting $\boldsymbol{c}_2^{(i)}$ and $\boldsymbol{c}_4^{(i)}$ starting at their corresponding intersections.

A function like Equation (41) that chooses the highest-scoring line for each span over γ is called an **envelope**, and can be used to compactly express the results we will obtain by rescoring $\boldsymbol{c}^{(i)}$ according to a particular γ (Figure 6a). After finding the envelope, for each line that participates in the envelope, we can calculate the sufficient statistics necessary for calculating the loss $\ell_{\text{error}}(\cdot)$ and error error$(\cdot)$. For example, given the envelope in Figure 6a, Figure 6b is an example of the sentence-wise loss with respect to γ.

The envelope shown in Equation (41) can also be viewed as the problem of finding a **convex hull** in computational geometry. A standard and efficient algorithm for finding a convex hull of multiple lines is the **sweep line algorithm** (Bentley and Ottmann 1979; Macherey et al. 2008) (see Figure 7). Here, we assume L is a set of the lines corresponding to the K translation candidates in $\boldsymbol{c}^{(i)}$, each line $l \in L$ is expressed as $\langle a(l), b(l), \gamma(l) \rangle$ with intercept $a(l) = a(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d})$, and slope $b(l) = b(\boldsymbol{f}^{(i)}, \boldsymbol{e}, \boldsymbol{d})$. Furthermore, we define $\gamma(l)$ as an intersection initialized to $-\infty$. SORTLINES(L) in Figure 3 sorts the lines in the order of their slope $b(l)$, and if two lines l_{k_1} have the same slope, l_{k_2} chooses the one with the larger intercept $a(l_{k_1}) > a(l_{k_2})$ and deletes the other. We next process the sorted set of lines L' ($|L'| \leq K$) in order of ascending slope (lines 4–18). If we assume H



(a) Envelope

(b) Loss

Figure 6
An example of line search in MERT.

```
1: procedure SweepLine(L = {⟨a(l^(k)), b(l^(k)), γ(l^(k))⟩}_{k=1}^K)
2:     H ← ∅, j ← 1
3:     L' ← SortLines(L)
4:     for l ∈ L' do
5:         if j > 1 then
6:             while j > 1 do
7:                 γ(l) ← (a(l) − a(H_{j−1})) / (b(H_{j−1}) − b(l))   ▷ Intersection of H_{j−1} and l
8:                 if γ(H_{j−1}) < γ(l) then                          ▷ l is in the envelope
9:                     break
10:                end if
11:                j ← j − 1
12:            end while
13:        end if
14:        if j = 1 then                                              ▷ The leftmost line
15:            γ(l) ← −∞
16:        end if
17:        H_j ← l, j ← j + 1
18:    end for
19:    return H_1^{j−1}
20: end procedure
```

Figure 7
The "sweep line" algorithm.

is the envelope expressed as the set of lines it contains, we find the line that intersects with line under consideration at the highest point (lines 6–12), and update the envelope H. As L contains at most K lines, H's size is also at most K.

Given a particular input sentence $\boldsymbol{f}^{(i)}$, its set of translation candidates $\boldsymbol{c}^{(i)}$, and the resulting envelope $H^{(i)}$, we can also define the set of intersections between lines in the envelope as $\gamma_1^{(i)} < \cdots < \gamma_j^{(i)} < \cdots < \gamma_{|H^{(i)}|}^{(i)}$. We also define $\Delta\ell_j^{(i)}$ to be the change in the loss function that occurs when we move from one span $[\gamma_{j-1}^{(i)}, \gamma_j^{(i)})$ to the next $[\gamma_j^{(i)}, \gamma_{j+1}^{(i)})$. If we first calculate the loss incurred when setting $\gamma = -\infty$, then process the spans in increasing order, keeping track of the difference $\Delta\ell_j^{(i)}$ incurred at each span boundary, it is possible to efficiently calculate the loss curve over all spans of γ.

In addition, whereas all explanation of line search to this point has focused on the procedure for a single sentence, by calculating the envelopes for each sentence in the data $1 \leq i \leq N$, and combining these envelopes into a single plane, it is relatively simple to perform this processing on the corpus level as well. It should be noted that for corpus-based evaluation measures such as BLEU, when performing corpus-level processing, we do not keep track of the change in the loss, but the change in the sufficient statistics required to calculate the loss for each sentence. In the case of BLEU, the sufficient statistics amount to n-gram counts c_n, n-gram matches m_n, and reference lengths r. We then calculate the loss curve $\ell_{\text{error}}(\cdot)$ for the entire corpus based on these sufficient statistics, and find a γ that minimizes Equation (17) based on this curve. By repeating this line search for each parameter until we can no longer obtain a decrease, it is possible to find a local minimum in the loss function, even for non-convex or non-differential functions.

5.1.3 MERT's Weaknesses and Extensions. Although MERT is widely used as the standard optimization procedure for MT, it also has a number of weaknesses, and a number of extensions to the MERT framework have been proposed to resolve these problems.

The first weakness of MERT is the **randomness** in the optimization process. Because each iteration of the training algorithm generally involves a number of random restarts, the results will generally change over multiple training runs, with the changes often being quite significant. Some research has shown that this randomness can be stabilized somewhat by improving the ability of the line-search algorithm to find a globally good solution by choosing random seeds more intelligently (Moore and Quirk 2008; Foster and Kuhn 2009) or by searching in directions that consider multiple features at once, instead of using the simple coordinate ascent as described in Figure 4 (Cer, Jurafsky, and Manning 2008). Orthogonally to actual improvement of the results, Clark et al. (2011) suggest that because randomness is a fundamental feature of MERT and other optimization algorithms for MT, it is better experimental practice to perform optimization multiple times, and report the resulting means and standard deviations over various optimization runs.

It is also possible to optimize the MERT objective using other optimization algorithms. For example, Suzuki, Duh, and Nagata (2011) present a method for using **particle swarm optimization**, a distributed algorithm where many "particles" are each associated with a parameter vector, and the particle updates its vector in a way such that it moves towards the current local and global optima. Another alternative optimization algorithm is Galley and Quirk's (2011) method for using **linear programming** to perform search for optimal parameters over more than one dimension, or all dimensions at a single time. However, as MERT remains a fundamentally computationally hard problem, this method takes large amounts of time for larger training sets or feature spaces.

It should be noted that instability in MERT is not entirely due to the fact that search is random, but also due to the fact that k-best lists are poor approximations of the whole space of possible translations. One way to improve this approximation is by performing MERT over an exponentially large number of hypotheses encoded in a translation lattice (Macherey et al. 2008) or hypergraph (Kumar et al. 2009). It is possible to perform MERT over these sorts of packed data structures by observing the fact that the envelopes used in MERT can be expressed as a **semiring** (Dyer 2010a; Sokolov and Yvon 2011), allowing for exact calculation of the full envelope for all hypotheses in a lattice or hypergraph using polynomial-time dynamic programming (the **forward algorithm** or **inside algorithm**, respectively). There has also been work to improve the accuracy of the k-best approximation by either sampling k-best candidates from the translation lattice (Chatterjee and Cancedda 2010), or performing forced decoding to find derivations that achieve the reference translation, and adding them to the k-best list (Liang, Zhang, and Zhao 2012).

The second weakness of MERT is that it has no concept of **regularization**, causing it to overfit the training data if there are too many features, and there have been several attempts to incorporate regularization to ameliorate this problem. Cer, Jurafsky, and Manning (2008) propose a method to incorporate regularization by not choosing the plateau in the loss curve that minimizes the loss itself, but choosing the point considering the loss values for a few surrounding plateaus, helping to avoid points that have a low loss but are surrounded by plateaus with higher loss. It is also possible to incorporate regularization into MERT-style line search using an SVM-inspired margin-based objective (Hayashi et al. 2009) or by using scale-invariant regularization methods such as L_0 or a scaled version of L_2 (Galley et al. 2013).

The final weakness of MERT is that it has computational problems when scaling to **large numbers of features**. When using only a standard set of 20 or so features, MERT is able to perform training in reasonable time, but the number of line searches, and thus time, required in Algorithm 4 scales linearly with the number of features. Thus training of hundreds of features is time-consuming, and there are no published results training standard MERT on thousands or millions of features. It should be noted, however, that Galley et al. (2013) report results for thousands of features by choosing intelligent search directions by calculating the gradient of expected BLEU, as explained in Section 5.5.2.

5.2 Gradient-Based Batch Optimization

In the previous section, MERT optimized a loss function that was exactly equivalent to the error function, which is not continuously differentiable and thus precludes the use of standard convex optimization algorithms used in other optimization problems. In contrast, other losses such as the softmax loss described in Section 3.2 and risk-based losses described in Section 3.3 are differentiable, allowing for the use of these algorithms for MT optimization (Smith and Eisner 2006; Blunsom and Osborne 2008).

Convex optimization is well covered in the standard machine learning literature, so we do not cover it in depth, but methods such as **conjugate gradient** (using first-order statistics) (Nocedal and Wright 2006) and the **limited-memory Broyden-Fletcher-Goldfarb-Shanno** method (using second-order statistics) (Liu and Nocedal 1989) are standard options for optimizing these losses. These methods are equally applicable when the loss is combined with a differentiable regularizer $\Omega(\boldsymbol{w})$, such as L_2 regularization. Using a non-differentiable regularizer such as L_1 makes optimization more difficult, but can be handled by other algorithms such as **orthant-wise limited-memory Quasi-Newton** (Andrew and Gao 2007).

In addition to the function being differentiable, if it is also convex we can be guaranteed that these algorithms will not get stuck in local optima and instead they will reach a globally optimal solution. In general, the softmax objective is convex if there is only one element in the oracle set $\boldsymbol{o}^{(i)}$, and not necessarily convex if there are multiple oracles. In the case of MT, as there are usually multiple translations $\boldsymbol{e}$ that minimize error$(\cdot)$, and multiple derivations $\boldsymbol{d}$ that result in the same translation $\boldsymbol{e}$, $\boldsymbol{o}^{(i)}$ will generally contain multiple members. Thus, we cannot be entirely certain that we will reach a global optimum.

5.3 Margin-Based Optimization

Minimizing the margin-based loss described in Section 3.4, possibly with the addition of a regularizer, is also a relatively standard problem in the machine learning literature. Methods to solve Equation (25) include **sequential minimization optimization** (Platt 1999), **dual coordinate descent** (Hsieh et al. 2008), as well as the **quadratic program** solvers used in standard SVMs (Joachims 1998).

It should also be noted that there have also been several attempts to apply margin-based online learning algorithms explained in Section 6.3, but in a batch setting where the whole training corpus is decoded before each iteration of optimization (Cherry and Foster 2012; Gimpel and Smith 2012). We will explain these methods in more detail later, but it should be noted that the advantage of using these methods in a batch

setting mainly lies in simplicity; for online learning it is often necessary to directly implement the optimization procedure within the decoder, whereas in a batch setting the implementation of the decoding and optimization algorithm can be performed separately.

5.4 Ranking and Linear Regression Optimization

The rank-based loss described in Section 3.5 is essentially the combination of multiple losses over binary decisions. These binary decisions can be solved using gradient-based or margin-based methods, and thus optimization itself can be solved with the algorithms described in the previous two sections. However, one important concern in this setting is training time. At the worst, the number of pairwise comparisons for any particular k-best list is $k(k-1)/2$, leading to unmanageably large amounts of time required for training.

One way to alleviate this problem is by randomly sampling a small number of these $k(k-1)/2$ hypotheses for use in optimization, which has been shown empirically to allow for increases in training speed without decreases in accuracy. For example, Hopkins and May (2011) describe a method dubbed **pairwise ranking optimization** that selects 5,000 pairs randomly for each sentence, and among these random pairs using the 50 with the largest difference in error for training the classifier. Other selection heuristics—for example, avoiding training on candidate pairs with overly different scores (Nakov, Guzmán, and Vogel 2013), or performing Monte Carlo sampling (Roth et al. 2010; Haddow, Arun, and Koehn 2011)—are also possible and potentially increase accuracy. Recently, there has also been a method proposed that uses an efficient ranking SVM formulation that alleviates the need for this sampling and explicitly performs ranking over all pairs (Dreyer and Dong 2015).

The mean squared error loss described in Section 3.6, which is similar to ranking loss in that it will prefer a proper ordering of the k-best list, is much easier to optimize. This loss can be minimized using standard techniques for solving least-squared-error linear regression (Press et al. 2007).

5.5 Risk Minimization

The risk-based loss in Section 3.3 is also differentiable and thus can be optimized by gradient-based methods. One thing to note here is that risk objective is highly non-convex, and thus some care must be taken to ensure that optimization does not fall into a local optimum. The motivation behind introducing the scaling parameter γ in Equation (22) is that it allows us to control the "peakiness" of the distribution, which can be useful for optimization. Putting this formally, we first define the entropy of $p_{\gamma,\boldsymbol{w}}(\cdot)$ as

$$H(p_{\gamma,\boldsymbol{w}}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{\langle \mathbf{e},\boldsymbol{d}\rangle \in \boldsymbol{c}^{(i)}} p_{\gamma,\boldsymbol{w}}(\boldsymbol{e},\boldsymbol{d}|\boldsymbol{f}^{(i)},\boldsymbol{c}^{(i)}) \log p_{\gamma,\boldsymbol{w}}(\boldsymbol{e},\boldsymbol{d}|\boldsymbol{f}^{(i)},\boldsymbol{c}^{(i)}) \tag{44}$$

When γ takes a small value this entropy will be high, indicating that the loss function is relatively smooth and less sensitive to local optima. Conversely, when $\gamma \to \infty$, the entropy becomes lower, and the loss function becomes more peaky with more local

optima. It has been noted that this fact can be used for effective optimization through the process of **deterministic annealing** (Sindhwani, Keerthi, and Chapelle 2006). In deterministic annealing, the parameter γ is not set as a hyperparameter, and instead the entropy $H(p_{\gamma,\boldsymbol{w}})$ is directly used as a regularization function during the optimization process (Smith and Eisner 2006; Li and Eisner 2009):

$$\underset{\gamma\in\mathbb{R}_{\geq 0},\boldsymbol{w}\in\mathbb{R}^M}{\arg\min}\ \ell_{\text{risk}}(F,E,C;\gamma,\boldsymbol{w}) - T\cdot H(p_{\gamma,\boldsymbol{w}}) + \lambda\Omega(\boldsymbol{w}) \tag{45}$$

In Equation (45), T is the **temperature**, which can either be set as a hyperparameter, or gradually decreased from ∞ to $-\infty$ (or 0) through a process of **cooling** (Smith and Eisner 2006). The motivation for cooling is that if we start with a large T, the earlier steps using a smoother function will allow us to approach the global optimum, and the later steps will allow us to approach the actual error function.

It should be noted that in Equation (24), and the discussion up to this point, we have been using not the corpus-based error, but the sentence-based error $\text{err}(\boldsymbol{e}^{(i)},\boldsymbol{e})$. There have also been attempts to make the risk minimization framework applicable to corpus-level error $\text{error}(\cdot)$, specifically BLEU. We will discuss two such methods.

5.5.1 Linear BLEU. **Linear BLEU** (Tromble et al. 2008) provides an approximation for corpus-level BLEU that can be divided among sentences. Linear BLEU uses a **Taylor expansion** to approximate the effect that the sufficient statistics of any particular sentence will have on corpus-level BLEU. We define r as the total length of the reference translations, c as the total length of the candidates, and c_n and m_n ($1\leq n\leq 4$) as the translation candidate's number of n-grams, and number of n-grams that match the reference respectively. Taking the equation for corpus-level BLEU (Papineni et al. 2002) and assuming that the n-gram counts are approximately equal for $1\leq n\leq 4$, we get the following approximation:

$$\log \text{BLEU} = \min\left\{0, 1-\frac{r}{c}\right\} + \frac{1}{4}\sum_{n=1}^{4}\log\frac{m_n}{c_n} \tag{46}$$

$$\approx \min\left\{0, 1-\frac{r}{c}\right\} + \frac{1}{4}\sum_{n=1}^{4}\log\frac{m_n}{c} \tag{47}$$

If we assume that when we add the sufficient statistics of a particular sentence $\boldsymbol{e}$, the corpus-level statistics change to r', c', c'_n, and m'_n, then we can express the change in BLEU in the logarithm domain as follows

$$\Delta\log\text{BLEU} = \log\text{BLEU}' - \log\text{BLEU} \tag{48}$$

$$\approx \frac{1}{4}\sum_{n=1}^{4}\log\frac{m'_n}{c'} - \frac{1}{4}\sum_{n=1}^{4}\log\frac{m_n}{c} \tag{49}$$

If we make the assumption that there is no change in the brevity penalty, $\Delta \log \text{BLEU}$ relies solely on m_n and c. $\Delta \log \text{BLEU}$ can then be approximated using a first-order Taylor expansion as follows:

$$\Delta \log \text{BLEU} \approx (c' - c) \left. \frac{\partial \log \text{BLEU}'}{\partial c'} \right|_{c'=c} + \sum_{n=1}^{4} (m'_n - m_n) \left. \frac{\partial \log \text{BLEU}'}{\partial m'_n} \right|_{m'_n = m_n} \tag{50}$$

$$= -\frac{c' - c}{c} + \frac{1}{4} \sum_{n=1}^{4} \frac{m'_n - m_n}{m_n} \tag{51}$$

As $c' - c$ is the length of $\boldsymbol{e}$, and $m'_n - m_n$ is the number of n-grams (g_n) in $\boldsymbol{e}$ that match the n-grams ($g_n^{(i)}$) in $\boldsymbol{e}^{(i)}$, the sentence-level error function $\text{err}_{\text{lBLEU}}(\cdot)$ for linear BLEU is

$$\text{err}_{\text{lBLEU}}(\boldsymbol{e}^{(i)}, \boldsymbol{e}) = \frac{|\boldsymbol{e}|}{c} - \frac{1}{4} \sum_{n=1}^{4} \frac{|\{g_n \in \boldsymbol{e}\} \cap \{g_n^{(i)} \in \boldsymbol{e}^{(i)}\}|}{m_n} \tag{52}$$

c and m_n are set to a fixed value (Tromble et al. 2008). For example, in the batch optimization algorithm of Figure 3 they can be calculated based on the k-best list generated prior to optimization.

5.5.2 Expectations of Sufficient Statistics. DeNero, Chiang, and Knight (2009) present an alternative method that calculates not the expectation of the error itself, but the expectation of the **sufficient statistics** used in calculating the error. In contrast to sentence-level approximations or formulations such as linear BLEU, the expectation of the sufficient statistics can be calculated directly on the corpus level. Because of this, by maximizing the evaluation derived by these expected statistics, it is possible to directly optimize for a corpus-level error, in a manner similar to MERT (Pauls, Denero, and Klein 2009).

When this is applied to BLEU in particular, this measure is often called **xBLEU** (Rosti et al. 2010, 2011) and the required sufficient statistics include n-gram counts and matched n-gram counts. We define the kth translation candidate in $\boldsymbol{c}^{(i)}$ as $\langle \boldsymbol{e}_k, \boldsymbol{d}_k \rangle$, its score as $s_{i,k} = \gamma \boldsymbol{w}^\top \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}_k, \boldsymbol{d}_k)$, and the probability in Equation (22) as

$$p_{i,k} = \frac{\exp(s_{i,k})}{\sum_{k'=1}^{K} \exp(s_{i,k'})} \tag{53}$$

Next, we define the expectation of the n-gram ($g_n \in \boldsymbol{e}_k$) frequency as $c_{n,i,k}$, the expectation of the number of n-gram matches as $m_{n,i,k}$, and the expectation of the reference length as $r_{i,k}$. These values can be calculated as:

$$c_{n,i,k} = |\{g_n \in \boldsymbol{e}_k\}| \cdot p_{i,k}$$

$$m_{n,i,k} = |\{g_n \in \boldsymbol{e}_k\} \cap \{g_n^{(i)} \in \boldsymbol{e}^{(i)}\}| \cdot p_{i,k}$$

$$r_{i,k} = |\boldsymbol{e}^{(i)}| \cdot p_{i,k}$$

It should be noted that although these equations apply to k-best lists, it is also possible to calculate statistics over lattices or forests using dynamic programming algorithms and tools such as the **expectation semiring** (Eisner 2002; Li and Eisner 2009).

xBLEU is calculated from these expected sufficient statistics:

$$\text{xBLEU} = \prod_{n=1}^{4} \left(\frac{\sum_{i=1}^{N} \sum_{k=1}^{K} m_{n,i,k}}{\sum_{i=1}^{N} \sum_{k=1}^{K} c_{n,i,k}} \right)^{\frac{1}{4}} \cdot \phi \left(1 - \frac{\sum_{i=1}^{N} \sum_{k=1}^{K} r_{i,k}}{\sum_{i=1}^{N} \sum_{k=1}^{K} c_{1,i,k}} \right) \tag{54}$$

where $\phi(x)$ is the brevity penalty. Compared to the risk minimization in Equation (24), we define our optimization problem as the maximization of xBLEU:

$$\ell_{\text{xBLEU}}(F, E, C; \gamma, \boldsymbol{w}) = -\text{xBLEU} \tag{55}$$

It is possible to calculate a gradient for xBLEU, allowing for optimization using gradient-based optimization methods, and we explain the full (somewhat involved) derivation in Appendix A.

6. Online Methods

In the batch learning methods of Section 5, the steps of decoding and optimization are performed sequentially over the entire training data. In contrast, **online learning** performs updates not after the whole corpus has been processed, but over smaller subsets of the training data deemed **mini-batches**. One of the major advantages of online methods is that updates are performed on a much more fine-grained basis—it is often the case that online methods converge faster than batch methods, particularly on larger data sets. On the other hand, online methods have the disadvantage of being harder to implement (they often must be implemented inside the decoder, whereas batch methods can be separate), and also generally being less stable (with sensitivity to the order in which the training data is processed or other factors).

In the online learning algorithm in Figure 8, from the training data $\langle F, E \rangle = \left\{ \langle \boldsymbol{f}^{(i)}, \boldsymbol{e}^{(i)} \rangle \right\}_{i=1}^{N}$ we first randomly choose a mini-batch consisting of K sentences of

procedure ONLINELEARN($\langle F, E \rangle = \left\{ \langle \boldsymbol{f}^{(i)}, \boldsymbol{e}^{(i)} \rangle \right\}_{i=1}^{N}$)
  $\boldsymbol{w}^{(1)} \leftarrow \emptyset$
  for $t \in \{1, \ldots, T\}$ **do**
    $\langle \tilde{F}^{(t)}, \tilde{E}^{(t)} \rangle \subseteq \langle F, E \rangle$ ▷ Sample ($|\langle \tilde{F}^{(t)}, \tilde{E}^{(t)} \rangle| = K$)
    $\tilde{C}^{(t)} \leftarrow \{\tilde{\boldsymbol{c}}^{(j)} = \emptyset\}_{j=1}^{K}$ ▷ k-best translation candidates
    for $\langle \tilde{\boldsymbol{f}}^{(j)}, \tilde{\boldsymbol{e}}^{(j)} \rangle \in \langle \tilde{F}^{(t)}, \tilde{E}^{(t)} \rangle$ **do**
      $\tilde{\boldsymbol{c}}^{(j)} \leftarrow \text{GEN}(\tilde{\boldsymbol{f}}^{(j)}, \boldsymbol{w}^{(t)})$ ▷ Decode with $\boldsymbol{w}^{(t)}$
    end for
    $\boldsymbol{w}^{(t+1)} \leftarrow \arg\min_{\boldsymbol{w} \in \mathbb{R}^M} \ell(\tilde{F}^{(t)}, \tilde{E}^{(t)}, \tilde{C}^{(t)}; \boldsymbol{w}) + \lambda \Omega(\boldsymbol{w})$ ▷ Optimization
  end for
  return $\boldsymbol{w}^{(T+1)}$
end procedure

Figure 8
Online Learning.

parallel data $\langle \tilde{F}^{(t)}, \tilde{E}^{(t)} \rangle = \left\{ \langle \tilde{\boldsymbol{f}}^{(j)}, \tilde{\boldsymbol{e}}^{(j)} \rangle \right\}_{j=1}^{K}$ (line 4). We then decode each source sentence $\tilde{\boldsymbol{f}}^{(j)}$ of the mini-batch and generate a k-best list (line 7), which is used in optimization (line 9). In contrast to the batch learning algorithm in Figure 3, we do not merge the k-bests from previous iterations. In addition, optimization is performed not over the entire data, but only the data $\langle \tilde{F}^{(t)}, \tilde{E}^{(t)} \rangle$ and its corresponding k-best, $\tilde{C}^{(t)}$. Like batch learning, within the online learning framework, there are a number of optimization algorithms and objective functions that can be used.

The first thing we must consider during online learning is that because we only optimize over the data in the mini-batch, it is not possible to directly optimize a corpus-level evaluation measure such as BLEU, and it is necessary to define an error function that is compatible with the learning framework (see Section 6.1). Once the error has been set, we can perform parameter updates according to a number of different algorithms including the perceptron (Section 6.2), MIRA (Section 6.3), AROW (Section 6.4), and stochastic gradient descent (SGD) (Section 6.5).

6.1 Approximating the Error

In online learning, parameters are updated not with respect to the entire training corpus, but with respect to a subset of data sampled from the corpus. This has consequences for the calculation of translation quality when using a corpus-level evaluation measure such as BLEU. For example, when choosing an oracle for oracle-based optimization methods, the oracles chosen when considering the entire corpus will be different from the oracles chosen when considering a mini-batch. In general, the amount of difference between the corpus-level and mini-batch level oracles will vary depending on the size of a mini-batch, with larger mini-batches providing a better approximation (Tan et al. 2013; Watanabe 2012). Thus, when using smaller batches, especially single sentences, it is necessary to use methods to approximate the corpus-level error function as covered in the next two sections.

6.1.1 Approximation with a Pseudo-Corpus. The first method to approximate the corpus-level evaluation measure relies on creating a **pseudo-corpus**, and using it to augment the statistics used in the mini-batch error calculation (Watanabe et al. 2007). Specifically, given the training data $\langle F, E \rangle = \left\{ \langle \boldsymbol{f}^{(i)}, \boldsymbol{e}^{(i)} \rangle \right\}_{i=1}^{N}$, we define its corresponding pseudo-corpus $\bar{E} = \left\{ \bar{\boldsymbol{e}}^{(i)} \right\}_{i=1}^{N}$. $\bar{E}$ could be, for example, either the 1-best translation candidate or the oracle calculated during the decoding step in line 7 of Figure 8. In the pseudo-corpus approximation, the sentence-level error for the translation candidate $\boldsymbol{e}'$ acquired by decoding the ith source sentence in the training data can be defined as the corpus-level error acquired when in $\bar{E}$, the ith sentence $\bar{\boldsymbol{e}}^{(i)}$ is replaced with $\boldsymbol{e}'$

$$\mathrm{err}_{\mathrm{pseudo}}(\boldsymbol{e}^{(i)}, \boldsymbol{e}') = \mathrm{error}\left(E, \{\bar{\boldsymbol{e}}^{(1)}, \ldots, \bar{\boldsymbol{e}}^{(i-1)}, \boldsymbol{e}', \bar{\boldsymbol{e}}^{(i+1)}, \ldots, \bar{\boldsymbol{e}}^{(N)}\}\right) \tag{56}$$

6.1.2 Approximation with Decay. When approximating the error function using a pseudo-corpus, it is necessary to remember translation candidates for every sentence in the corpus. In addition, the size of differences in the sentence-level error becomes dependent on the number of other sentences in the corpus, making it necessary to perform scaling of the error, particularly for max-margin methods (Watanabe et al. 2007). As an alternative method that alleviates these problems, there has also been a method

proposed that remembers a single set of sufficient statistics for the whole corpus, and upon every update forces these statistics to **decay** according to some criterion (Chiang, Marton, and Resnik 2008; Chiang, Knight, and Wang 2009; Chiang 2012).

We define $\bar{\mathbf{s}}$ as the sufficient statistics necessary to calculate a particular evaluation measure. For example, in the case of BLEU, this would be a particular translation candidate's counts of the number of n-grams, the number of matched n-grams, and the reference length. When evaluating the error for a candidate e' of the ith sentence in the training data, we first decay the sufficient statistics

$$\bar{\mathbf{s}} \leftarrow \nu \times \bar{\mathbf{s}} \tag{57}$$

where $\nu < 1$ is the amount of decay, taking a value such as 0.9 or 0.99. Next, based on these sufficient statistics, we calculate the error of each candidate for the sentence by summing the sufficient statistics of the sentence with the decayed sufficient statistics $\bar{\mathbf{s}}$. For example, if we want to calculate BLEU using $\text{stat}_{\text{BLEU}}(e^{(i)}, e')$ and the ith training sentence $\langle f^{(i)}, e^{(i)} \rangle$, if BLEU$(\cdot)$ is a function calculating BLEU from a particular set of sufficient statistics, we can use the following equation:

$$\text{err}_{\text{bleu}'}(e^{(i)}, e') = 1 - \text{BLEU}(\bar{\mathbf{s}} + \text{stat}_{\text{BLEU}}(e^{(i)}, e')) \tag{58}$$

After performing an update for a particular sentence, $\bar{\mathbf{s}}$ is then updated with the statistics from the 1-best hypothesis found during decoding. When the training data is large, this function will place more emphasis on the recently generated examples, forgetting the older ones.

6.2 The Perceptron

The most simple algorithm for online learning is the **perceptron algorithm** (shown in Figure 9), which, as its name suggests, optimizes the perceptron loss of Section 3.4. The most central feature of the algorithm is that when the 1-best and oracle translations

procedure PERCEPTRON($\langle F, E \rangle = \left\{ \langle f^{(i)}, e^{(i)} \rangle \right\}_{i=1}^{N}$)
  $w^{(1)} \leftarrow 0$
  for $t \in \{1 \dots T\}$ **do**
    $\langle f, e \rangle \sim \langle F, E \rangle$ ▷ Assume $K = 1$
    $\tilde{c} \leftarrow \text{GEN}(f, w^{(t)})$ ▷ Decode with $w^{(t)}$
    $\langle \hat{e}, \hat{d} \rangle \in \tilde{c}$ ▷ 1-best candidate
    $\langle e^*, d^* \rangle \in \tilde{o} \subseteq \tilde{c}$ ▷ Oracle candidate
    if $\langle e^*, d^* \rangle \neq \langle \hat{e}, \hat{d} \rangle$ **then**
      $w^{(t+1)} \leftarrow w^{(t)} + h(f, e^*, d^*) - h(f, \hat{e}, \hat{d})$ ▷ Update
    end if
  end for
  return $w^{(T+1)}$ or $\frac{1}{T} \sum_{t=2}^{T+1} w^{(t)}$
end procedure

Figure 9
The perceptron algorithm.

differ, we update the parameters at line 9. Because the gradient of the perceptron loss in Equation (31) with respect to $\boldsymbol{w}$ is

$$\frac{1}{K}\sum_{i=1}^{K} -\Delta\boldsymbol{h}(\tilde{\boldsymbol{f}}^{(i)}, \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)}, \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)}) = \frac{1}{K}\sum_{i=1}^{K} -\left(\boldsymbol{h}(\tilde{\boldsymbol{f}}^{(i)}, \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)}) - \boldsymbol{h}(\tilde{\boldsymbol{f}}^{(i)}, \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)})\right) \quad (59)$$

this algorithm can also be viewed as updating the parameters based on this gradient, making the parameters for oracle $\langle \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)} \rangle$ stronger, and the parameters for the mistaken translation $\langle \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)} \rangle$ weaker.

In line 12, we return the final parameters to be used in translation. The most straightforward approach here is to simply return the parameters resulting from the final iteration of the perceptron training, but in a popular variant called the **averaged perceptron**, we instead use the average of the parameters over all iterations in training (Collins 2002). This averaging helps reduce overfitting of sentences that were viewed near the end of the training process, and is known to improve robustness to unknown data, resulting in higher translation accuracy (Liang et al. 2006).

6.3 MIRA

In line 9 of the perceptron algorithm in Figure 9, the parameters are updated using the gradient with respect to $\boldsymbol{w}^{(t)}$ (see Equation (59)). This update has the advantage of being simple and being guaranteed to converge when the data is linearly separable, but it is common for MT to handle feature sets that do not allow for linear separation of the 1-best and oracle hypotheses, resulting in instability in learning. The **margin infused relaxed algorithm** (MIRA) (Crammer and Singer 2003; Crammer et al. 2006) is another online learning algorithm designed to help reduce these problems of instability. The update in MIRA follows the same Equation (59), but also adds an additional term that prevents the parameters $\boldsymbol{w}^{(t)}$ from varying largely from their previous values[8]

$$\ell_{\text{MIRA}}(\tilde{F}, \tilde{E}, \tilde{C}; \boldsymbol{w}, \boldsymbol{w}^{(t)}) = \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}^{(t)}\|_2^2 + \frac{\lambda_{\text{MIRA}}}{K}\sum_{i=1}^{K} \max\left\{0, \Delta\,\text{err}(\tilde{\boldsymbol{e}}^{(i)}, \boldsymbol{e}^{*(i)}, \hat{\boldsymbol{e}}^{(i)}) - \boldsymbol{w}^{\top}\Delta\boldsymbol{h}(\tilde{\boldsymbol{f}}^{(i)}, \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)}, \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)})\right\} \quad (60)$$

It should be noted that although this is defined as a loss, it is dependent on the parameters at the previous time step, in contrast to the losses in Section 3.

In line 9 of Figure 8 (or when $K = 1$, line 9 of Figure 9), the next parameters $\boldsymbol{w}^{(t+1)}$ are chosen to minimize Equation (60) (Watanabe et al. 2007; Chiang, Marton, and Resnik 2008; Chiang, Knight, and Wang 2009). λ_{MIRA} is a hyperparameter that controls the

8 The algorithm presented here is actually the PA-I algorithm (Crammer et al. 2006) applied to structured learning, and for historical reasons most research in parsing, MT, and other areas refer to this algorithm not as PA-I, but MIRA (McDonald, Crammer, and Pereira 2005; Watanabe et al. 2007).

amount of fitting to the data, with larger values indicating a stronger fit. Intuitively, the MIRA objective contains an error-minimizing term similar to that of the perceptron, but also contains a regularization term with respect to the change in parameters compared to $\boldsymbol{w}^{(t)}$, preferring smaller changes in parameters. When $K > 1$, this equation can be formulated using Lagrange multipliers and solved using a quadratic programming solver similar to that used in SVMs. When $K = 1$, we can simply use the following update formula

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha^{(t)} \Delta \boldsymbol{h}(\tilde{\boldsymbol{f}}, \boldsymbol{e}^*, \boldsymbol{d}^*, \hat{\boldsymbol{e}}, \hat{\boldsymbol{d}}) \tag{61}$$

$$\alpha^{(t)} = \min \left\{ \lambda_{\text{MIRA}}, \frac{\Delta \operatorname{err}(\tilde{\boldsymbol{e}}, \boldsymbol{e}^*, \hat{\boldsymbol{e}}) - \boldsymbol{w}^\top \Delta \boldsymbol{h}(\tilde{\boldsymbol{f}}, \boldsymbol{e}^*, \boldsymbol{d}^*, \hat{\boldsymbol{e}}, \hat{\boldsymbol{d}})}{\| \Delta \boldsymbol{h}(\tilde{\boldsymbol{f}}, \boldsymbol{e}^*, \boldsymbol{d}^*, \hat{\boldsymbol{e}}, \hat{\boldsymbol{d}}) \|^2} \right\} \tag{62}$$

The amount of update is proportional to the difference in loss between hypotheses, but when the difference in the features between the oracle and incorrect hypotheses is small, the features that are different will be subject to an extremely large update. The function of the parameter λ_{MIRA} is to control these over-aggressive updates. It should also be noted that when we set $\alpha^{(t)} = 1$, MIRA reduces to the perceptron algorithm.

6.4 AROW

One of the problems often pointed out with MIRA is that it is overagressive, mainly because it attempts to classify the current training example correctly according to Equation (60), even when the training example is an outlier or includes noise. One way to reduce the effect of noise is through the use of **adaptive regularization of weights** (AROW) (Crammer, Kulesza, and Dredze 2009; Chiang 2012). AROW is based on a similar concept to MIRA, but instead of working directly on the weight vector $\boldsymbol{w}$, it defines a Gaussian distribution $\mathcal{N}(\boldsymbol{w}, \Sigma)$ over the weights. The covariance matrix Σ is usually assumed to be diagonal, and each variance term in Σ functions as a sort of learning rate for its corresponding weight, with weights of higher variance being updated more widely, and weights with lower variance being updated less widely.

For notational simplicity, we first define $x^{(i)} = \Delta \boldsymbol{h}(\tilde{\boldsymbol{f}}^{(i)}, \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)}, \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)})$, after which we can express the AROW loss as follows:

$$\ell_{\text{AROW}}(\tilde{F}, \tilde{E}, \tilde{C}; \boldsymbol{w}, \Sigma, \boldsymbol{w}^{(t)}, \Sigma^{(t)}) = \operatorname{KL}(\mathcal{N}(\boldsymbol{w}, \Sigma) || \mathcal{N}(\boldsymbol{w}^{(t)}, \Sigma^{(t)})) + \frac{1}{K} \sum_{i=1}^{K} \left(\lambda_{\text{MIRA}} \max \left\{ 0, \Delta \operatorname{err}(\tilde{\boldsymbol{e}}^{(i)}, \boldsymbol{e}^{*(i)}, \hat{\boldsymbol{e}}^{(i)}) - \boldsymbol{w}^\top x^{(i)} \right\} + \frac{\lambda_{\text{var}}}{2} x^{\top(i)} \Sigma x^{(i)} \right) \tag{63}$$

where $\operatorname{KL}(\cdot)$ is the Kullback-Leibler divergence. The KL divergence term here plays a similar role to the parameter norm in MIRA, preventing large changes in the weight distributions from update to update, and the first term within the summation is the same loss term as MIRA. The main difference lies in the second term within the summation, which penalizes variance matrices that have large values for features seen in the hypotheses to be updated. This has the effect of decreasing the variance, and thus the learning rate, for frequently updated features, preventing large and aggressive moves of weights for features that already have been seen often and thus can be considered relatively reliable. In the case of $K = 1$ and where $\lambda_{\text{MIRA}} = \lambda_{\text{var}}$, the AROW update that

minimizes this loss consists of the following updates of $\boldsymbol{w}$ and Σ (Crammer, Kulesza, and Dredze 2009):

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha^{(t+1)}\Sigma^{(t)}x^{(i)} \tag{64}$$

$$\Sigma^{(t+1)} = \Sigma^{(t)} - \beta^{(t+1)}\Sigma^{(t)}x^{(i)}x^{\top(i)}\Sigma^{(t)} \tag{65}$$

$$\alpha^{(t+1)} = \max\left(0, 1 - \boldsymbol{w}^{\top(t)}x^{(i)}\right)\beta^{(t+1)} \tag{66}$$

$$\beta^{(t+1)} = \frac{1}{x^{\top(i)}\Sigma_{t-1}x^{(i)} + 1/2\lambda_{\text{var}}}. \tag{67}$$

6.5 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a gradient-based online algorithm for optimizing differentiable losses, possibly with the addition of L_2 regularization $\Omega_2(\boldsymbol{w})$, or L_1 regularization $\Omega_1(\boldsymbol{w})$. As SGD relies on gradients, it can be thought of as an alternative to the gradient-based batch algorithms in Section 5.2. Compared with batch algorithms, SGD requires less memory and tends to converge faster, but requires more care (particularly with regard to the selection of a learning rate) to ensure that it converges to a good answer.

In SGD, like with the perceptron algorithm and MIRA, in line 9 of Figure 8 the parameters are updated according to the gradient $\Delta\ell(\cdot) + \lambda\Delta\Omega(\cdot)$ with respect to $\boldsymbol{w}^{(t)}$ (Bottou 1998):

$$\eta^{(t+1)} \leftarrow \text{update}(\eta^{(t)}) \tag{68}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta^{(t+1)}\left\{\Delta\ell(\tilde{F}^{(t)}, \tilde{E}^{(t)}, \tilde{C}^{(t)}; \boldsymbol{w}^{(t)}) + \lambda\Delta\Omega(\boldsymbol{w}^{(t)})\right\} \tag{69}$$

where $\eta^{(t)} > 0$ is the **learning rate**, which is generally initialized to a value $\eta^{(1)}$ and gradually reduced according to a function update$(\cdot)$ as learning progresses. One standard method for updating $\eta^{(t)}$ according to the following formula

$$\eta^{(t+1)} \leftarrow \frac{\eta^{(1)}}{1 + t/T} \tag{70}$$

allows for a guarantee of convergence (Collins et al. 2008). In Equation (69), the parameters are updated and we obtain $\boldsymbol{w}^{(t+1)}$. Within this framework, in the perceptron algorithm $\eta^{(t)}$ is set to a fixed value, and in MIRA the amount of update changes for every mini-batch. SGD-style online gradient-based methods have been used in translation for optimizing risk-based (Gao and He 2013), ranking-based (Watanabe 2012; Green et al. 2013), and other (Tillmann and Zhang 2006) objectives. When the regularization term $\Omega(\boldsymbol{w})$ is not differentiable, such as L_1 regularization, it is a common practice to use **forward-backward splitting** (FOBOS) (Duchi and Singer 2009; Green et al. 2013) in which the optimization is performed in two steps:

$$\boldsymbol{w}^{(t+\frac{1}{2})} \leftarrow \boldsymbol{w}^{(t)} - \eta^{(t+1)}\Delta\ell(\tilde{F}^{(t)}, \tilde{E}^{(t)}, \tilde{C}^{(t)}; \boldsymbol{w}^{(t)}) \tag{71}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}^{(t+\frac{1}{2})}\|_2^2 + \eta^{(t+1)}\lambda\Omega(\boldsymbol{w}) \tag{72}$$

First, we perform updates without considering the regularization term in Equation (71). Second, the regularization term is applied in Equation (72), which balances regularization and proximity to $\boldsymbol{w}^{(t+\frac{1}{2})}$. As an alternative to FOBOS, it is possible to use **dual averaging**, which keeps track of the average of previous gradients and optimizes for these along with the full regularization term (Xiao 2010).

An alternative to SGD is **adaptive gradient** (AdaGrad) (Duchi, Hazan, and Singer 2011; Green et al. 2013) updates. The motivation behind AdaGrad is similar to that of AROW (Section 6.4), using second-order covariance statistics Σ to adjust the learning rate of individual parameters based on their update frequency. If we define the SGD gradient as $x = \Delta\ell(\tilde{F}^{(t)}, \tilde{E}^{(t)}, \tilde{C}^{(t)}; \boldsymbol{w}^{(t)}) + \lambda\Delta\Omega(\boldsymbol{w}^{(t)})$ for notational simplicity, the update rule for AdaGrad can be expressed as follows

$$\Sigma^{-1(t+1)} \leftarrow \Sigma^{-1(t)} + xx^\top \tag{73}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta\Sigma^{1/2(t)}x \tag{74}$$

Like AROW, it is common to use a diagonal covariance matrix, and each time an update is performed the variance for the updated features decreases, reducing the overall learning rate for more commonly updated features. It should be noted that as the update of each feature is automatically controlled by the covariance matrix, there is no need to decay η as is necessary in SGD.

7. Large-Scale Optimization

Up to this point, we have generally given a mathematical or algorithmic explanation of the various optimization methods, and placed a smaller emphasis on factors such as training efficiency. In traditional optimization settings for MT where we optimize only a small number of weights for dense features on a training set of around 1,000 sentences, efficiency is often less of a concern. However, when trying to move to larger sets of sparse features, 1,000 sentences of training data is simply not enough to robustly estimate the parameters, and larger training sets become essential. When moving to larger training sets, parallelization of both the decoding process and the optimization process becomes essential. In this section, we outline the methods that can be used to perform parallelization, greatly increasing the efficiency of training. As parallelization in MT has seen wider use with respect to online learning methods, we will start with a description of online methods and touch briefly upon batch methods afterwards.

7.1 Large-Scale Online Optimization

Within the online learning framework, it is possible to improve the efficiency of learning through parallelization (McDonald, Hall, and Mann 2010). An example of this is shown in Figure 10, where the training data $\langle F, E\rangle = \left\{\langle \boldsymbol{f}^{(i)}, \boldsymbol{e}^{(i)}\rangle\right\}_{i=1}^{N}$ is split into S **shards** (line 2), learning is performed locally over each shard $\langle F_s, E_s\rangle$, and the S sets of parameters $\boldsymbol{w}_s$ acquired through local learning are combined according to a function mix$(\cdot)$, a process called **parameter mixing** (line 6). This can be considered an instance of the **MapReduce** (Dean and Ghemawat 2008) programming model, where each Map assigns shards to S CPUs and performs training, and Reduce combines the resulting parameters $\boldsymbol{w}_s$.

In the training algorithm of Figure 10, because parameters are learned locally on each shard, it is not necessarily guaranteed that the parameters are optimized for the

1: **procedure** PARALLELLEARN($\langle F, E \rangle$)
2: Split $\langle F, E \rangle$ into S shards $\{\langle F_1, E_1 \rangle, \ldots, \langle F_S, E_S \rangle\}$
3: **for** $s \in \{1, \ldots, S\}$ **parallel do** ▷ Run shards in parallel
4: $\boldsymbol{w}_s \leftarrow$ ONLINELEARN($\langle F_s, E_s \rangle$)
or BATCHLEARN($\langle F_s, E_s \rangle$) ▷ Local learning
5: **end for**
6: $\boldsymbol{w} \leftarrow \text{mix}(\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S\})$ ▷ Parameter mixing
7: **return** $\boldsymbol{w}$
8: **end procedure**

Figure 10
Parallel learning.

data as a whole. In addition, it is also known that some divisions of the data can lead to contradictions between the parameters (McDonald, Hall, and Mann 2010). Because of this, when performing distributed online learning, it is common to perform parameter mixing several times throughout the training process, which allows the separate shards to share information and prevents contradiction between the learned parameters. Based on the timing of the update, these varieties of mixing are called **synchronous update** and **asynchronous update**.

7.1.1 Synchronous Update. In the online learning algorithm with synchronous update shown in Figure 11, learning is performed independently over each shard $\langle F_s, E_s \rangle (1 \leq s \leq S)$. The difference between this and Figure 10 lies in the fact that learning is performed T' times, with each iteration initialized with the parameters $\boldsymbol{w}^{(t)}$ from the previous iteration (line 7).

After the local learning finishes, the parameters are mixed in line 8 (Watanabe et al. 2007; McDonald, Hall, and Mann 2010; Watanabe 2012). In the mixing function it is most common to use the average of the parameters

$$\boldsymbol{w}^{(t+1)} \leftarrow \sum_{s=1}^{S} \mu_s \bar{\boldsymbol{w}}_s^{(t+1)} \tag{75}$$

1: **procedure** SYNCHRONOUSONLINELEARN($\langle F, E \rangle$)
2: Split $\langle F, E \rangle$ into S fragments $\{\langle F_1, E_1 \rangle, \ldots, \langle F_S, E_S \rangle\}$
3: $\boldsymbol{w}^{(1)} \leftarrow 0$ ▷ Initialize parameters
4: **for** $s \in \{1, \ldots, S\}$ **parallel do** ▷ Run shards in parallel
5: **for** $t \in \{1, \ldots, T\}$ **do**
6: $\bar{\boldsymbol{w}}_s^{(t)} \leftarrow \boldsymbol{w}^{(t)}$ ▷ Copy parameters to each shard
7: $\bar{\boldsymbol{w}}_s^{(t+1)} \leftarrow$ ONLINELEARN($\langle F_s, E_s \rangle, \bar{\boldsymbol{w}}_s^{(t)}, T'$)
▷ Local learning on each shard
8: $\boldsymbol{w}^{(t+1)} \leftarrow \text{mix}\left(\{\bar{\boldsymbol{w}}_1^{(t+1)}, \ldots, \bar{\boldsymbol{w}}_S^{(t+1)}\}\right)$ ▷ Mix parameters
9: **end for**
10: **end for**
11: **return** $\boldsymbol{w}^{(T+1)}$ or $\frac{1}{T}\sum_{t=2}^{T+1} \boldsymbol{w}^{(t)}$
12: **end procedure**

Figure 11
Online learning with synchronous update.

where $\sum_{s=1}^{S} \mu_s = 1$. As μ_s, it is possible to use a uniform distribution, or a weight proportional to the number of online updates performed at each shard (McDonald, Hall, and Mann 2010; Simianer, Riezler, and Dyer 2012). It should be noted that this algorithm can be considered a variety of the MapReduce framework, allowing for relatively straightforward implementation using parallel processing infrastructure such as Hadoop (Eidelman 2012).

Simianer, Riezler, and Dyer (2012) propose another method for mixing parameters that, instead of averaging at each iteration, chooses to preserve only the parameters that have been learned over all shards, and sets all the remaining parameters to zero, allowing for a simple sort of feature selection. In particular, we define a $S \times M$ matrix that combines the parameters $\bar{w}_s^{(t+1)}$ at each shard as $\bar{w}^{(t+1)} = \left[\bar{w}_1^{(t+1)} | \ldots | \bar{w}_S^{(t+1)}\right]^\top$, takes the L_2 norm of each matrix column, and averages the columns with high norm values while setting the rest to zero.

7.1.2 Asynchronous Update. While parallel learning with synchronous update is guaranteed to converge (McDonald, Hall, and Mann 2010), parameter mixing only occurs after all the data has been processed, leading to inefficiency over large data sets. To fix this problem, asynchronous update sends information about parameter updates to each shard asynchronously, allowing the parameters to be updated more frequently, resulting in faster learning (Chiang, Marton, and Resnik 2008; Chiang, Knight, and Wang 2009).

The algorithm for learning with asynchronous update is shown in Figure 12. With the data $\langle F_s, E_s \rangle$ $(1 \leq s \leq S)$ split into S pieces, each shard performs T iterations of training by sampling a mini-batch (line 7), translating each sentence (line 10), and performing optimization on the mini-batch level (line 12).

1: **procedure** AsynchronousOnlineLearn($\langle F, E \rangle$)
2: Split $\langle F, E \rangle$ into S fragments $\{\langle F_1, E_1 \rangle, \ldots, \langle F_S, E_S \rangle\}$
3: $\boldsymbol{w}^{(1)} \leftarrow 0$ ▷ Initialize parameters
4: **for** $s \in \{1, \ldots, S\}$ **parallel do** ▷ Run s shards in parallel
5: **for** $t \in \{1, \ldots, T\}$ **do**
6: $\bar{w}_s^{(t)} \leftarrow \text{copy}_{\text{async}}\left(\boldsymbol{w}^{(t)}\right)$ ▷ Copy parameters to each shard
7: $\langle \tilde{F}_s^{(t)}, \tilde{E}_s^{(t)} \rangle \subseteq \langle F_s, E_s \rangle$ ▷ Sample ($|\langle \tilde{F}_s^{(t)}, \tilde{E}_s^{(t)} \rangle| = K$)
8: $\tilde{C}_s^{(t)} \leftarrow \{\tilde{\boldsymbol{c}}_s^{(j)} = \emptyset\}_{j=1}^{K}$ ▷ k-best candidates
9: **for** $\langle \tilde{\boldsymbol{f}}_s^{(j)}, \tilde{\boldsymbol{e}}_s^{(j)} \rangle \in \langle \tilde{F}_s^{(t)}, \tilde{E}_s^{(t)} \rangle$ **do**
10: $\tilde{\boldsymbol{c}}_s^{(j)} \leftarrow \text{GEN}(\tilde{\boldsymbol{f}}_s^{(j)}, \bar{w}_s^{(t)})$ ▷ Decode with $\bar{w}_s^{(t)}$
11: **end for**
12: $\bar{w}_s^{(t+1)} \leftarrow \arg\min_{\bar{w}_s \in \mathbb{R}^M} \ell(\tilde{F}_s^{(t)}, \tilde{E}_s^{(t)}, \tilde{C}_s^{(t)}; \bar{w}_s) + \lambda\Omega(\bar{w}_s)$ ▷ Optimize
13: $\boldsymbol{w}^{(t+1)} \leftarrow \text{mix}_{\text{async}}\left(\{\bar{w}_1^{(t+1)}, \ldots, \bar{w}_S^{(t+1)}\}\right)$ ▷ Mix parameters
14: **end for**
15: **end for**
16: **return** $\boldsymbol{w}^{(T+1)}$ or $\frac{1}{T}\sum_{t=2}^{T+1} \boldsymbol{w}^{(t)}$
17: **end procedure**

Figure 12
Online learning with asynchronous update.

Compared with the synchronous algorithm in Figure 11, the parameter mixing operator $\text{mix}_{\text{async}}(\cdot)$ in line 13 sends the result of the mini-batch level update

$$\Delta\bar{w}_s^{(t+1)} = \bar{w}_s^{(t+1)} - \bar{w}_s^{(t)} \tag{76}$$

to each shard $s'(s' \neq s, 1 \leq s' \leq S)$. It should be noted that because the number of dimensions M in the parameter vector is extremely large, and each mini-batch will only update a small fraction of these parameters, by only sending the parameters that have actually changed in each mini-batch update we can greatly increase the efficiency of the training. In operator $\text{copy}_{\text{async}}(\cdot)$ of line 6, the learner receives the update from all the other shards and mixes them together to acquire the full update vector

$$\bar{w}_s^{(t)} \leftarrow \sum_{s'=1}^{S} \Delta\bar{w}_{s'}^{(t)} \tag{77}$$

It should be noted that at mixing time, there is no need to wait for the update vectors from all of the other shards; update can be performed with only the update vectors that have been received at the time. Because of this, each shard will not necessarily be using parameters that reflect all the most recent updates performed on other shards. However, as updates are performed much more frequently than in synchronous update, it is easier to avoid local optima, and learning tends to converge faster. It can also be shown that (under some conditions) the amount of accuracy lost by this delay in update is bounded (Zinkevich, Langford, and Smola 2009; Recht et al. 2011).

7.2 Large-Scale Batch Optimization

Compared with online learning, within the batch optimization framework, parallelization is usually straightforward. Often, decoding takes the majority of time required for the optimization process, and because the parameters will be identical for each sentence in the decoding run (Figure 3, line 6), decoding can be parallelized trivially. The process of parallelizing optimization itself depends slightly on the optimization algorithm, but is generally possible to achieve in a number of ways.

The first, and simplest, method for parallelization is the parallelization of **optimization runs**. The most obvious example of this is MERT, where random restarts are required. Each of the random restarts is completely independent, so it is possible to run these on different nodes, and finally check which run achieved the best accuracy and use that result.

Another more fine-grained method for parallelization, again most useful for MERT, is the parallelization of **search directions**. In the loop starting at Figure 4, line 6, MERT performs line search in several different directions, each one being independent of the others. Each of these line searches can be performed in parallel, and the direction allowing for the greatest gain in accuracy is chosen when all threads have completed.

A method that is applicable to a much broader array of optimization methods is the parallelization of **calculation of sufficient statistics**. In this approach, like in Section 7.1.1, we first split the data into shards $\langle F_s, E_s \rangle (1 \leq s \leq S)$. Then, over these shards we calculate the sufficient statistics necessary to perform a parameter update. For example, in MERT these sufficient statistics would consist of the envelope for each of the potential search directions. In gradient based methods, the sufficient statistics would

consist of the gradient calculated with respect to only the data on the shard. Finally, when all threads have finished calculating these statistics, a master thread combines the statistics from each shard, either by merging the envelopes, or by adding the gradients.

8. Other Topics in MT Optimization

In this section we cover several additional topics in optimization for MT, including nonlinear models (Section 8.1), optimization for a particular domain or test set (Section 8.2), and the interaction between evaluation measures (Section 8.3) or search (Section 8.4) and optimization.

8.1 Non-Linear Models

Note that up until this point, all models that we have considered calculate the scores for translation hypotheses according to a linear model, where the score is calculated according to the dot product of the features and weights shown in Equation (1). However, linear models are obviously limited in their expressive power, and a number of works have attempted to move beyond linear combinations of features to nonlinear combinations.

In general, most nonlinear models for machine learning can be applied to MT as well, with one major caveat. Specifically, the efficiency of the decoding process largely relies on the feature locality assumption mentioned in Section 2.3. Unfortunately, the locality assumption breaks down when moving beyond a simple linear scoring function, and overcoming this problem is the main obstacle to applying nonlinear models to MT (or structured learning in general). A number of countermeasures to this problem exist:

Reranking: The most simple and commonly used method for incorporating nonlinearity, or other highly nonlocal features that cannot be easily incorporated in search, is through the use of reranking (Shen, Sarkar, and Och 2004). In this case, a system optimized using a standard linear model is used to create a k-best list of outputs, and this k-best list is then reranked using the nonlinear model (Nguyen, Mahajan, and He 2007; Duh and Kirchhoff 2008). Because we are now only dealing with fully expanded hypotheses, scoring becomes trivial, but reranking also has the major downsides of potentially missing useful hypotheses not included in the k-best list,[9] and requiring time directly proportional to the size of the k-best list.

Local Nonlinearity: Another possibility is to first use a nonlinear function to calculate local features, which are then used as part of the standard linear model (Liu et al. 2013). Alternatively, it is possible to treat feature-value pairs as new binary features (Clark, Dyer, and Lavie 2014). In this case, all effects of nonlinearity are resolved before the search actually begins, allowing for the use of standard and efficient search algorithms. On the other hand, it is not possible to incorporate non-local features into the nonlinear model.

Improved Search Techniques: Although there is no general-purpose solution to incorporating nonlinear models into search, for some particular models it is possible to perform search in a way that allows for incorporation of nonlinearities. For example, **ensemble decoding** has been used with stacking-based models (Razmara

9 This problem can be ameliorated somewhat by ensuring that there is sufficient diversity in the n-best list (Gimpel et al. 2013).

and Sarkar 2013), and it has been shown that the search space can be simplified to the extent that kernel functions can be calculated efficiently (Wang, Shawe-Taylor, and Szedmak 2007).

Once the problems of search have been solved, a number of actual learning techniques can be used to model nonlinear scoring functions. One of the most popular examples of nonlinear functions are those utilizing kernels, and methods applied to MT include kernel-like functions over the feature space such as the Parzen window, binning, and Gaussian kernels (Nguyen, Mahajan, and He 2007), or the *n*-spectrum string kernel for finding associations between the source and target strings (Wang, Shawe-Taylor, and Szedmak 2007). Neural networks are another popular method for modeling nonlinearities, and it has been shown that neural networks can effectively be used to calculate new local features for MT (Liu et al. 2013). Methods such as boosting or stacking, which combine together multiple parameterizations of the translation model, have been incorporated through reranking (Duh and Kirchhoff 2008; Lagarda and Casacuberta 2008; Duan et al. 2009; Sokolov, Wisniewski, and Yvon 2012b), or ensemble decoding (Razmara and Sarkar 2013). Regression decision trees have also been introduced as a method for inducing nonlinear functions, incorporated through history-based search algorithms (Turian, Wellington, and Melamed 2006), or by using the trees to induce features local to the search state (Toutanova and Ahn 2013).

8.2 Domain-Dependent Optimization

One widely acknowledged feature of machine learning problems in general is that the parameters are sensitive to the domain of the data, and by optimizing the parameters with data from the target domain it is possible to achieve gains in accuracy. In machine translation, this is also very true, although much of the work on domain adaptation has focused on adapting the model learning process prior to explicit optimization towards an evaluation measure (Koehn and Schroeder 2007). However, there are a few works on optimization-based domain adaptation in MT, as we will summarize subsequently.

One relatively simple way of performing domain adaptation is by selecting a **subset** of the training data that is similar to the data that we want to translate (Li et al. 2010). This can be done by selecting sentences that are similar to our test corpus, or even selecting adaptation data for each individual test sentence (Liu et al. 2012). If no parallel data exist in the target domain, it has also been shown that first automatically translating data from the source to the target language or vice versa, then using this data for optimization and model training is also helpful (Ueffing, Haffari, and Sarkar 2007; Li et al. 2011; Zhao et al. 2011) In addition, in a computer-assisted translation scenario, it is possible to reflect post-edited translations back into the optimization process as new in-domain training data (Mathur, Mauro, and Federico 2013; Denkowski, Dyer, and Lavie 2014).

Once adaptation data have been chosen, it is necessary to decide how to use the data. The most straightforward way is to simply use these in-domain data in optimization, but if the data set is small it is preferable to combine both in- and out-of-domain data to achieve more robust parameter estimates. This is essentially equivalent to the standard domain-adaptation problem in machine learning, and in the context of MT there have been methods proposed to perform Bayesian adaptation of probabilistic models (Sanchis-Trilles and Casacuberta 2010), and online update using ultraconservative algorithms (Liu et al. 2012). This can be extended to cover multiple target domains using multi-task learning (Cui et al. 2013).

Finally, it has been noted that when optimizing a few dense parameters, it is useful to make the distinction between in-domain translation (when the model training data matches the test domain) and cross-domain translation (when the model training data mismatches the test domain). In cross-domain translation, fewer long rules will be used, and translation probabilities will be less reliable, and the parameters must change accordingly to account for this (Pecina, Toral, and van Genabith 2012). It has also been shown that building TMs for several domains and tuning the parameters to maximize translation accuracy can improve MT accuracy on the target domain (Haddow 2013). Another option for making the distinction between in-domain and out-of-domain data is by firing different features for in-domain and out-of-domain training data, allowing for the learning of different weights for different domains (Clark, Lavie, and Dyer 2012).

8.3 Evaluation Measures and Optimization

In the entirety of this article, we have assumed that optimization for MT aims to reduce MT error defined using an evaluation measure, generally BLEU. However, as mentioned in Section 2.5, evaluation of MT is an active research field, and there are many alternatives in addition to BLEU. Thus, it is of interest whether changing the measure used in optimization can affect the overall quality of the translations achieved, as measured by human evaluators.

There have been a few comprehensive studies on the effect of the metric used in optimization on human assessments of the generated translations (Cer, Manning, and Jurafsky 2010; Callison-Burch et al. 2011). These studies showed the rather surprising result that despite the fact that other evaluation measures had proven superior to BLEU with regards to post facto correlation with human evaluation, a BLEU-optimized system proved superior to systems tuned using other metrics. Since this result, however, there have been other reports stating that systems optimized using other metrics such as TESLA (Liu, Dahlmeier, and Ng 2011) and MEANT (Lo et al. 2013) achieve superior results to BLEU-optimized systems.

There have also been attempts to directly optimize not automatic, but **human evaluation measures** of translation quality (Zaidan and Callison-Burch 2009). However, the cost of performing this sort of human-in-the-loop optimization is prohibitive, so Zaidan and Callison-Burch (2009) propose a method that re-uses partial hypotheses in evaluation. Saluja, Lane, and Zhang (2012) also propose a method for incorporating binary good/bad input into optimization, with the motivation that this sort of feedback is easier for human annotators to provide than generating new reference sentences.

Finally, there is also some work on optimizing **multiple evaluation metrics** at one time. The easiest way to do so is to simply use the linear interpolation of two or more metrics as the error function (Dyer et al. 2009; He and Way 2009; Servan and Schwenk 2011):

$$\text{error}(E, \hat{E}) = \sum_{i=1}^{L} \rho_i \, \text{error}_i(E, \hat{E}) \tag{78}$$

where L is the number of error functions, and ρ_i is a manually set interpolation coefficient for its respective error function. There are also more sophisticated methods based on the idea of optimizing towards Pareto-optimal hypotheses (Duh et al.

2012), which achieve errors lower than all other hypotheses on at least one evaluation measure,

$$\text{pareto}(E, \mathcal{E}) = \{\hat{E} \in \mathcal{E} : \forall_{E' \in \mathcal{E}} \exists_i \, \text{error}_i(E, \hat{E}) < \text{error}_i(E, E')\} \tag{79}$$

To incorporate this concept of Pareto optimality into optimization, the Pareto-optimal set is defined on the sentence level, and ranking loss (Section 3.5) is used to ensure that the Pareto-optimal hypotheses achieve a higher score than those that are not Pareto optimal. This method has also been extended to take advantage of ensemble decoding, where multiple parameter settings are used simultaneously in decoding (Sankaran, Sarkar, and Duh 2013).

8.4 Search and Optimization

As mentioned in Section 2.4, because MT decoders perform approximate search, they may make search errors and not find the hypothesis that achieves the highest model score. There have been a few attempts to consider this fact in the optimization process.

For example, in the perceptron algorithm of Section 6.2 it is known that the convergence guarantees of the structured perceptron no longer hold when using approximate search. The first method that can be used to resolve this problem is the **early updating** strategy (Collins and Roark 2004; Cowan, Kučerová, and Collins 2006). The early updating strategy is a variety of bold updates, where the decoder output $\boldsymbol{e}^{*(i)}$ must be exactly equal to the reference $\boldsymbol{e}^{(i)}$. Decoding proceeds as normal, but the moment the correct hypothesis $\boldsymbol{e}^{(i)}$ can no longer be produced by any hypothesis in the search space (i.e., a search error has occurred), search is stopped and update is performed using only the partial derivation. The second method is the **max-violation perceptron** (Huang, Fayong, and Guo 2012; Yu et al. 2013). In the max-violation perceptron, forced decoding is performed to acquire a derivation $\langle \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)} \rangle$ that can exactly reproduce the correct output $\boldsymbol{e}^{(i)}$, and update is performed at the point when the score of a partial hypothesis $\langle \hat{\boldsymbol{e}}^{(i)}, \hat{\boldsymbol{d}}^{(i)} \rangle$ exceeds that of the partial hypothesis $\langle \boldsymbol{e}^{*(i)}, \boldsymbol{d}^{*(i)} \rangle$ by the greatest margin (the point of "maximum violation").

Search-aware tuning (Liu and Huang 2014) is a method that is able to consider search errors using an arbitrary optimization method. It does so by defining an evaluation measure for not only full sentences, but also partial derivations that occur during the search process, and optimizes parameters for k-best lists of partial derivations.

Finally, there has also been some work on optimizing features not of the model itself, but parameters of the search process, using the downhill simplex algorithm (Chung and Galley 2012). Using this method, it is possible to adjust the beam width, distortion penalty, or other parameters that actually affect the size and shape of the derivation space, as opposed to simply rescoring hypotheses within it.

9. Conclusion

In this survey article, we have provided a review of the current state-of-the-art in machine translation optimization, covering batch optimization, online optimization, expansions to large scale data, and a number of other topics. While these optimization algorithms have already led to large improvements in machine translation accuracy, the task of MT optimization is, as stated in the Introduction, an extremely hard one that is far from solved.

Table 2
Percent of WMT systems using each optimization method, including all systems, or systems that achieved the best results on at least one language pair.

	WMT 2011		WMT 2012		WMT 2013		WMT 2014	
	All	Best	All	Best	All	Best	All	Best
MERT	80	100	79	100	68	25	63	50
MIRA	0	0	0	0	20	75	27	50
Ranking	0	0	4	0	8	0	5	0
Softmax	3	0	0	0	0	0	0	0
Risk	3	0	0	0	0	0	5	0
None	17	0	17	0	4	0	0	0

The utility of an optimization algorithm can be viewed from a number of perspectives. The final accuracy achieved is, of course, one of the most important factors, but speed, scalability, ease of implementation, final resulting model size, and many other factors play an important role. We can assume that the algorithms being used outside of the context of research on optimization itself are those that satisfy these criteria in some way. Although it is difficult to exactly discern exactly which algorithms are seeing the largest amount of use (industrial SMT systems rarely disclose this sort of information publicly), one proxy for this is to look at systems that performed well on shared tasks such as the Workshop on Machine Translation (WMT) (Bojar et al. 2014). In Table 2 we show the percentage of WMT systems using each optimization algorithm for the past four years, both including all systems, and systems that achieved the highest level of human evaluation in the resource-constrained setting for at least one language pair. From these statistics we can see that even after over ten years, MERT is still the dominant optimization algorithm. However, starting in WMT 2013, we can see a move to systems based on MIRA, and to a lesser extent ranking, particularly in the most competitive systems.

In these systems, the preferred choice of an optimization algorithm seems to be MERT when using up to 20 features, and MIRA when using a large number of features (up to several hundred). There are fewer examples of systems using large numbers of features (tens of thousands, or millions) in actual competitive systems, with a few exceptions (Dyer et al. 2009; Neidert et al. 2014; Wuebker et al. 2014). In the case when a large number of sparse features are used, it is most common to use a softmax or risk-based objective and gradient-based optimization algorithms, often combining the features into summary features and performing a final tuning pass with MERT.

The fact that algorithms other than MERT are seeing adoption in competitive systems for shared tasks is a welcome sign for the future of MT optimization research. However, there are still many open questions in the field, a few of which can be outlined here:

Stable Training with Millions of Features: At the moment, there is still no stable training recipe that has been widely proven to effectively optimize millions of features. Finding an algorithm that gives consistent improvements in this setting is perhaps the largest open problem in MT optimization.

Evaluation Measures for Optimization: Although many evaluation measures show consistent improvements in correlation with human evaluation scores over BLEU

when used to evaluate the output of existing MT systems, there are few results that show that systems optimized with evaluation measures other than BLEU achieve consistent improvements in human evaluation scores.

Better Training/Utilization of Nonlinear Scoring Functions: Nonlinear functions using neural networks have recently achieved large improvements in a number of areas of natural language processing and machine learning. Better methods to incorporate these sort of nonlinear scoring functions into MT is a highly promising direction, but will require improvements in both learning the scoring functions and correctly incorporating these functions into MT decoding.

Resolving these, and many other, open questions will likely be among the top priorities of MT optimization research in the years to come.

Appendix A: Derivation for xBLEU Gradients

In this appendix, we explain in detail how to derive a gradient for the xBLEU objective in Equation (54), which has not been described completely in previous work.

First, we consider the brevity penalty

$$\phi(x) = \min\{1, \exp(x)\} \tag{A.1}$$

We can see that it is not differentiable, which precludes the use of gradient-based algorithms. To remedy this problem, Rosti et al. (2010) propose the use of the following differentiable approximation for the brevity penalty.

$$\varphi(x) = \frac{\exp(x) - 1}{1 + \exp(10000x)} + 1 \tag{A.2}$$

Using Equation (54) and the approximated brevity penalty in Equation (A.2), we can express xBLEU $= \exp(P) \cdot B$, leading to the following equation

$$P = \frac{1}{4} \sum_{n=1}^{4} \left(\log \sum_{i=1}^{N} \sum_{k=1}^{K} m_{n,i,k} - \log \sum_{i=1}^{N} \sum_{k=1}^{K} c_{n,i,k} \right) \tag{A.3}$$

$$B = \varphi\,(1 - R) \tag{A.4}$$

$$R = \frac{\sum_{i=1}^{N} \sum_{k=1}^{K} r_{i,k}}{\sum_{i=1}^{N} \sum_{k=1}^{K} c_{1,i,k}} \tag{A.5}$$

Taking the derivative of xBLEU with respect to $\boldsymbol{w}$, we get

$$\frac{\partial\, \text{xBLEU}}{\partial \boldsymbol{w}} = \sum_{i=1}^{N} \sum_{k=1}^{K} \frac{\partial\, \text{xBLEU}}{\partial \log p_{i,k}} \frac{\partial \log p_{i,k}}{\partial \boldsymbol{w}} \tag{A.6}$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{k'=1}^{K} \frac{\partial\, \text{xBLEU}}{\partial \log p_{i,k}} \frac{\partial \log p_{i,k}}{\partial s_{i,k'}} \frac{\partial s_{i,k'}}{\partial \boldsymbol{w}} \tag{A.7}$$

Thus,

$$\frac{\partial s_{i,k'}}{\partial \boldsymbol{w}} = \gamma \cdot \boldsymbol{h}(\boldsymbol{f}^{(i)}, \boldsymbol{e}_{k'}, \boldsymbol{d}_{k'}) \tag{A.8}$$

$$\frac{\partial \log p_{i,k}}{\partial s_{i,k'}} = \delta(k,k') - p_{i,k'} \tag{A.9}$$

and

$$\frac{\partial \,\text{xBLEU}}{\partial \log p_{i,k}} = \exp(P) \cdot \frac{\partial B}{\partial \log p_{i,k}} + \exp(P) \cdot \frac{\partial P}{\partial \log p_{i,k}} \cdot B \tag{A.10}$$

Additionally, the following equation holds:

$$\frac{\partial B}{\partial \log p_{i,k}} = \frac{\partial B}{\partial 1-R} \frac{\partial 1-R}{\partial \log p_{i,k}} \tag{A.11}$$

$$= \varphi'(1-R) \cdot (-R) \cdot \left(\frac{r_{i,k}}{\sum_{i'=1}^{N} \sum_{k'=1}^{K} r_{i',k'}} - \frac{c_{1,i,k}}{\sum_{i'=1}^{N} \sum_{k'=1}^{K} c_{1,i',k'}} \right) \tag{A.12}$$

$$\frac{\partial P}{\partial \log p_{i,k}} = \frac{1}{4} \sum_{n=1}^{4} \frac{m_{n,i,k}}{\sum_{i'=1}^{N} \sum_{k'=1}^{K} m_{n,i',k'}} - \frac{c_{n,i,k}}{\sum_{i'=1}^{N} \sum_{k'=1}^{K} c_{n,i',k'}} \tag{A.13}$$

After calculating this gradient, it is possible to optimize this according to standard gradient-based methods. However, like MR using sentence-level evaluation mentioned in Section 5.5, the evaluation measure is not convex, and the same precautions need to be taken to avoid falling into local optima.

References

Andrew, Galen and Jianfeng Gao. 2007. Scalable training of L1-regularized log-linear models. In *Proceedings of ICML*, pages 33–40, Corvalis, OR.

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

Bazrafshan, Marzieh, Tagyoung Chung, and Daniel Gildea. 2012. Tuning as linear regression. In *Proceedings of HLT/NAACL*, pages 543–547, Montréal.

Bentley, J. L. and T. A. Ottmann. 1979. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 28(9):643–647.

Berger, Adam L., Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39–71.

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

Blunsom, Phil and Miles Osborne. 2008. Probabilistic inference for machine translation. In *Proceedings of EMNLP*, pages 215–223, Honolulu, HI.

Bojar, Ondrej, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of WMT*, pages 12–58, Baltimore, MD.

Bottou, Léon. 1998. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, pages 9–42, Cambridge, UK.

Brown, Peter F., Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of

statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.
Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of ICML*, pages 89–96, Bonn.
Callison-Burch, Chris, Philipp Koehn, Christof Monz, and Omar Zaidan. 2011. Findings of the 2011 workshop on statistical machine translation. In *Proceedings of WMT*, pages 22–64, Edinburgh.
Cao, Zhe, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of ICML*, pages 129–136, Corvalis, OR.
Cer, Daniel, Dan Jurafsky, and Christopher D. Manning. 2008. Regularization and search for minimum error rate training. In *Proceedings of WMT*, pages 26–34, Columbus, OH.
Cer, Daniel, Christopher D. Manning, and Daniel Jurafsky. 2010. The best lexical metric for phrase-based statistical MT system optimization. In *Proceedings of HLT/NAACL*, pages 555–563, Los Angeles, CA.
Chappelier, Jean-Cédric and Martin Rajman. 1998. A generalized CYK algorithm for parsing stochastic CFG. In *Proceedings of the 1st Workshop on Tabulation in Parsing and Deduction*, pages 133–137, Paris.
Chatterjee, Samidh and Nicola Cancedda. 2010. Minimum error rate training by sampling the translation lattice. In *Proceedings of EMNLP*, pages 606–615, Cambridge, MA.
Chen, Stanley F. and Ronald Rosenfeld. 1999. A Gaussian prior for smoothing maximum entropy models. Technical report CMU-CS-99-108, DTIC Document. Carnegie Mellon School of Computer Science.
Cherry, Colin and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proceedings of HLT/NAACL*, pages 427–436, Montréal, Canada.
Chiang, David. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.
Chiang, David. 2012. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13:1159–1187.
Chiang, David, Kevin Knight, and Wei Wang. 2009. 11,001 new features for statistical machine translation. In *Proceedings of HLT/NAACL*, pages 218–226, Boulder, CO.
Chiang, David, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proceedings of EMNLP*, pages 224–233, Honolulu, HI.
Chung, Tagyoung and Michel Galley. 2012. Direct error rate minimization for statistical machine translation. In *Proceedings of WMT*, pages 468–479, Montréal.
Clark, Jonathan H., Chris Dyer, and Alon Lavie. 2014. Locally non-linear learning for statistical machine translation via discretization and structured regularization. *Transactions of the Association for Computational Linguistics*, 2(1):393–404.
Clark, Jonathan H., Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of ACL/HLT*, pages 176–181, Portland, OR.
Clark, Jonathan H., Alon Lavie, and Chris Dyer. 2012. One system, many domains: Open-domain statistical machine translation via feature augmentation. In *Proceedings of AMTA*, San Diego, CA.
Collins, Michael. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, PA.
Collins, Michael, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L. Bartlett. 2008. Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *Journal of Machine Learning Research*, 9:1775–1822.
Collins, Michael and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, pages 111–118, Barcelona.
Cowan, Brooke, Ivona Kučerová, and Michael Collins. 2006. A discriminative model for tree-to-tree translation. In *Proceedings of EMNLP*, pages 232–241, Sydney.
Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.
Crammer, Koby, Alex Kulesza, and Mark Dredze. 2009. Adaptive regularization of weight vectors. In *Proceedings of NIPS*, pages 414–422, Vancouver.

Crammer, Koby and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Cui, Lei, Xilun Chen, Dongdong Zhang, Shujie Liu, Mu Li, and Ming Zhou. 2013. Multi-domain adaptation for SMT using multi-task learning. In *Proceedings of EMNLP*, pages 1055–1065, Seattle, WA.

Dean, Jeffrey and Sanjay Ghemawat. 2008. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

DeNero, John, David Chiang, and Kevin Knight. 2009. Fast consensus decoding over translation forests. In *Proceedings of ACL/IJCNLP*, pages 567–575, Singapore.

Denkowski, Michael, Chris Dyer, and Alon Lavie. 2014. Learning from post-editing: Online model adaptation for statistical machine translation. In *Proceedings of EACL*, pages 395–404, Gothenburg.

Dreyer, Markus and Yuanzhe Dong. 2015. APRO: All-pairs ranking optimization for MT tuning. In *Proceedings of NAACL*, pages 1018–1023, Denver, CO.

Dreyer, Markus and Daniel Marcu. 2012. Hyter: Meaning-equivalent semantics for translation evaluation. In *Proceedings of HLT/NAACL*, pages 162–171, Montréal.

Duan, Nan, Mu Li, Tong Xiao, and Ming Zhou. 2009. The Feature Subspace method for SMT system combination. In *Proceedings of EMNLP*, pages 1096–1104, Singapore.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Duchi, John and Yoram Singer. 2009. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934.

Duh, Kevin and Katrin Kirchhoff. 2008. Beyond log-linear models: Boosted minimum error rate training for n-best re-ranking. In *Proceedings of ACL/HLT: Short Papers*, pages 37–40, Columbus, OH.

Duh, Kevin, Katsuhito Sudoh, Xianchao Wu, Hajime Tsukada, and Masaaki Nagata. 2012. Learning to translate with multiple objectives. In *Proceedings of ACL*, pages 1–10, Jeju Island.

Dyer, Chris. 2010a. *A Formal Model of Ambiguity and its Applications in Machine Translation*. Ph.D. thesis, University of Maryland.

Dyer, Chris. 2010b. Two monolingual parses are better than one (synchronous parse). In *Proceedings of HLT/NAACL*, pages 263–266, Los Angeles, CA.

Dyer, Chris, Hendra Setiawan, Yuval Marton, and Philip Resnik. 2009. The University of Maryland statistical machine translation system for the Fourth Workshop on Machine Translation. In *Proceedings of WMT*, pages 145–149, Athens.

Eidelman, Vladimir. 2012. Optimization strategies for online large-margin learning in machine translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 477–486, Montréal.

Eidelman, Vladimir, Yuval Marton, and Philip Resnik. 2013. Online relative margin maximization for statistical machine translation. In *Proceedings of ACL*, pages 1116–1126, Sofia.

Eisner, Jason. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of ACL*, pages 1–8, Philadelphia, PA.

Flanigan, Jeffrey, Chris Dyer, and Jaime Carbonell. 2013. Large-scale discriminative training for statistical machine translation using held-out line search. In *Proceedings of HLT/NAACL*, pages 248–258, Atlanta, GA.

Foster, George and Roland Kuhn. 2009. Stabilizing minimum error rate training. In *Proceedings of WMT*, pages 242–249, Athens.

Freund, Yoav, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969.

Galley, Michel and Christopher D. Manning. 2008. A simple and effective hierarchical phrase reordering model. In *Proceedings of EMNLP*, pages 848–856, Honolulu, HI.

Galley, Michel and Chris Quirk. 2011. Optimal search for minimum error rate training. In *Proceedings of EMNLP*, pages 38–49, Edinburgh.

Galley, Michel, Chris Quirk, Colin Cherry, and Kristina Toutanova. 2013. Regularized minimum error rate training. In *Proceedings of EMNLP*, pages 1948–1959, Seattle, WA.

Gao, Jianfeng and Xiaodong He. 2013. Training MRF-based phrase translation models using gradient ascent. In *Proceedings of HLT/NAACL*, pages 450–459, Atlanta, GA.

Gesmundo, Andrea and James Henderson. 2011. Heuristic search for non-bottom-up tree structure prediction. In *Proceedings of EMNLP*, pages 899–908, Edinburgh.

Gimpel, Kevin, Dhruv Batra, Chris Dyer, and Gregory Shakhnarovich. 2013. A systematic exploration of diversity in machine translation. In *Proceedings of EMNLP*, pages 1100–1111, Seattle, WA.

Gimpel, Kevin and Noah A. Smith. 2009. Feature-rich translation by quasi-synchronous lattice parsing. In *Proceedings of EMNLP*, pages 219–228, Singapore.

Gimpel, Kevin and Noah A. Smith. 2012. Structured ramp loss minimization for machine translation. In *Proceedings of HLT/NAACL*, pages 221–231, Montréal.

Graham, Yvette, Timothy Baldwin, Alistair Moffat, and Justin Zobel. 2014. Is machine translation getting better over time? In *Proceedings of EACL*, pages 443–451, Gothenburg.

Green, Spence, Daniel Cer, and Christopher Manning. 2014. An empirical comparison of features and tuning for phrase-based machine translation. In *Proceedings of WMT*, pages 466–476, Baltimore, MD.

Green, Spence, Sida Wang, Daniel Cer, and Christopher D. Manning. 2013. Fast and adaptive online training of feature-rich translation models. In *Proceedings of ACL*, pages 311–321, Sofia.

Green, Spence, Sida I. Wang, Jason Chuang, Jeffrey Heer, Sebastian Schuster, and Christopher D. Manning. 2014. Human effort and machine learnability in computer aided translation. In *Proceedings of EMNLP*, pages 1225–1236, Doha.

Haddow, Barry. 2013. Applying pairwise ranked optimization to improve the interpolation of translation models. In *Proceedings of HLT/NAACL*, pages 342–347, Atlanta, GA.

Haddow, Barry, Abhishek Arun, and Philipp Koehn. 2011. SampleRank training for phrase-based machine translation. In *Proceedings of WMT*, pages 261–271, Edinburgh.

Hasan, Saša, Richard Zens, and Hermann Ney. 2007. Are very large N-best lists useful for SMT? In *Proceedings of HLT/NAACL*, pages 57–60, Rochester, NY.

Hayashi, Katsuhiko, Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2009. Structural support vector machines for log-linear approach in statistical machine translation. In *Proceedings of IWSLT*, pages 144–151, Tokyo.

He, Xiaodong and Li Deng. 2012. Maximum expected BLEU training of phrase and lexicon translation models. In *Proceedings of ACL*, pages 292–301, Jeju Island.

He, Yifan and Andy Way. 2009. Improving the objective function in minimum error rate training. In *Proceedings of MT Summit*, pages 238–245, Ottawa.

Herbrich, Ralf, Thore Graepel, and Klaus Obermayer. 1999. Support vector learning for ordinal regression. In *Proceedings of ICANN*, pages 97–102, Edinburgh.

Hopkins, Mark and Jonathan May. 2011. Tuning as ranking. In *Proceedings of EMNLP*, pages 1352–1362, Edinburgh.

Hsieh, Cho-Jui, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. 2008. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of ICML*, pages 408–415, Helsinki.

Huang, Liang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of ACL*, pages 144–151, Prague.

Huang, Liang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of HLT/NAACL*, pages 142–151, Montréal.

Huang, Liang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proceedings of AMTA*, pages 66–73, Cambridge, MA.

Joachims, Thorsten. 1998. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. Springer, New York.

Klein, Dan and Christopher D. Manning. 2004. Parsing and hypergraphs. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*. Kluwer Academic Publishers, Norwell, MA, pages 351–372.

Koehn, Philipp. 2010. *Statistical Machine Translation*. Cambridge University Press, New York, NY.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL: Demo and Poster Sessions*, pages 177–180, Prague.

Koehn, Philipp, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of HLT/NAACL*, pages 48–54, Edmonton.

Koehn, Philipp and Josh Schroeder. 2007. Experiments in domain adaptation for

statistical machine translation. In *Proceedings of WMT*, pages 224–227, Prague.

Kumar, Shankar, Wolfgang Macherey, Chris Dyer, and Franz Och. 2009. Efficient minimum error rate training and minimum Bayes-risk decoding for translation hypergraphs and lattices. In *Proceedings of ACL/IJCNLP*, pages 163–171, Singapore.

Lagarda, Antonio and Francisco Casacuberta. 2008. Applying boosting to statistical machine translation. In *Proceedings of EAMT*, pages 88–96, Hamburg.

Leusch, Gregor, Evgeny Matusov, and Hermann Ney. 2008. Complexity of finding the BLEU-optimal hypothesis in a confusion network. In *Proceedings of EMNLP*, pages 839–847, Honolulu, HI.

Li, Mu, Yinggong Zhao, Dongdong Zhang, and Ming Zhou. 2010. Adaptive development data selection for log-linear model in statistical machine translation. In *Proceedings of COLING*, pages 662–670, Beijing.

Li, Zhifei and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of EMNLP*, pages 40–51, Singapore.

Li, Zhifei and Sanjeev Khudanpur. 2009. Forest reranking for machine translation with the perceptron algorithm. In J. Olive, C. Christianson, & J. McCary, editors, *Handbook of Natural Language Processing and Machine Translation*. Springer, pages 226–236.

Li, Zhifei, Ziyuan Wang, Jason Eisner, Sanjeev Khudanpur, and Brian Roark. 2011. Minimum imputed-risk: Unsupervised discriminative training for machine translation. In *Proceedings of EMNLP*, pages 920–929, Edinburgh.

Liang, Huashen, Min Zhang, and Tiejun Zhao. 2012. Forced decoding for minimum error rate training in statistical machine translation. *Journal of Computational Information Systems*, 8(2):861–868.

Liang, Percy, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proceedings of COLING/ACL*, pages 761–768, Sydney.

Lin, Chin-Yew and Franz Josef Och. 2004. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of COLING*, pages 501–507, Geneva.

Liu, Chang, Daniel Dahlmeier, and Hwee Tou Ng. 2011. Better evaluation metrics lead to better machine translation. In *Proceedings of EMNLP*, pages 375–384, Edinburgh.

Liu, Dong C. and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528.

Liu, Lemao, Hailong Cao, Taro Watanabe, Tiejun Zhao, Mo Yu, and Conghui Zhu. 2012. Locally training the log-linear model for SMT. In *Proceedings of EMNLP/CoNLL*, pages 402–411, Jeju Island.

Liu, Lemao and Liang Huang. 2014. Search-aware tuning for machine translation. In *Proceedings of EMNLP*, pages 1942–1952, Doha.

Liu, Lemao, Taro Watanabe, Eiichiro Sumita, and Tiejun Zhao. 2013. Additive neural networks for statistical machine translation. In *Proceedings of ACL*, pages 791–801, Sofia.

Liu, Lemao, Tiejun Zhao, Taro Watanabe, Hailong Cao, and Conghui Zhu. 2012. Expected error minimization with ultraconservative update for SMT. In *Proceedings of COLING: Posters*, pages 723–732, Mumbai.

Liu, Lemao, Tiejun Zhao, Taro Watanabe, and Eiichiro Sumita. 2013. Tuning SMT with a large number of features via online feature grouping. In *Proceedings of IJCNLP*, pages 279–285, Nagoya.

Lo, Chi-kiu, Karteek Addanki, Markus Saers, and Dekai Wu. 2013. Improving machine translation by training against an automatic semantic frame based evaluation metric. In *Proceedings of ACL: Short Papers*, pages 375–381, Sofia.

Lopez, Adam. 2008. Statistical machine translation. *ACM Computing Surveys*, 40(3):1–49.

Macherey, Wolfgang, Franz Och, Ignacio Thayer, and Jakob Uszkoreit. 2008. Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of EMNLP*, pages 725–734, Honolulu, HI.

Marton, Yuval and Philip Resnik. 2008. Soft syntactic constraints for hierarchical phrased-based translation. In *Proceedings of ACL/HLT*, pages 1003–1011, Columbus, OH.

Mathur, Prashant, Cettolo Mauro, and Marcello Federico. 2013. Online learning approaches in computer assisted translation. In *Proceedings of WMT*, pages 301–308, Sofia.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98, Ann Arbor, MI.

McDonald, Ryan, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *Proceedings of HLT/NAACL*, pages 456–464, Los Angeles, CA.

Moore, Robert C. and Chris Quirk. 2008. Random restarts in minimum error rate training for statistical machine translation. In *Proceedings of COLING*, pages 585–592, Manchester.

Nakov, Preslav, Francisco Guzman, and Stephan Vogel. 2012. Optimizing for sentence-level BLEU+1 yields short translations. In *Proceedings of COLING*, pages 1979–1994, Mumbai.

Nakov, Preslav, Francisco Guzmán, and Stephan Vogel. 2013. A tale about pro and monsters. In *Proceedings of ACL: Short Papers*, pages 12–17, Sofia.

Neidert, Julia, Sebastian Schuster, Spence Green, Kenneth Heafield, and Christopher Manning. 2014. Stanford University's submissions to the WMT 2014 translation task. In *Proceedings of WMT*, pages 150–156, Baltimore, MD.

Nguyen, Patrick, Milind Mahajan, and Xiaodong He. 2007. Training non-parametric features for statistical machine translation. In *Proceedings of WMT*, pages 72–79, Prague.

Nocedal, Jorge and Stephen J. Wright. 2006. *Conjugate Gradient Methods*. Springer, New York.

Och, Franz Josef. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167, Sapporo.

Och, Franz Josef and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of ACL*, pages 295–302, Philadelphia, PA.

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

Papineni, K. A. 1999. Discriminative training via linear programming. In *Proceedings of ICASSP*, pages 561–564, Phoenix, AZ.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of ACL*, pages 311–318, Philadelphia, PA.

Pauls, Adam, John Denero, and Dan Klein. 2009. Consensus training for consensus decoding in machine translation. In *Proceedings of EMNLP*, pages 1418–1427, Singapore.

Pecina, Pavel, Antonio Toral, and Josef van Genabith. 2012. Simple and effective parameter tuning for domain adaptation of statistical machine translation. In *Proceedings of COLING*, pages 2209–2224, Mumbai.

Peitz, Stephan, Arne Mauser, Joern Wuebker, and Hermann Ney. 2012. Forced derivations for hierarchical machine translation. In *Proceedings of COLING: Posters*, pages 933–942, Mumbai.

Platt, John C. 1999. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods*. MIT Press, Cambridge, MA, pages 185–208.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY.

Razmara, Majid and Anoop Sarkar. 2013. Stacking for statistical machine translation. In *Proceedings of ACL: Short Papers*, pages 334–339, Sofia.

Recht, Benjamin, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of NIPS*, pages 693–701, Vancouver.

Rosti, Antti-Veikko, Bing Zhang, Spyros Matsoukas, and Richard Schwartz. 2010. BBN system description for WMT10 system combination task. In *Proceedings of WMT*, pages 321–326, Uppsala.

Rosti, Antti-Veikko, Bing Zhang, Spyros Matsoukas, and Richard Schwartz. 2011. Expected BLEU training for graphs: BBN system description for WMT11 system combination task. In *Proceedings of WMT*, pages 159–165, Edinburgh.

Roth, Benjamin, Andrew McCallum, Marc Dymetman, and Nicola Cancedda. 2010. Machine translation using overlapping alignments and SampleRank. In *Proceedings of AMTA*, Denver, CO.

Saluja, Avneesh, Ian Lane, and Ying Zhang. 2012. Machine translation with binary feedback: A large-margin approach. In *Proceedings of AMTA*, San Diego, CA.

Sanchis-Trilles, Germán and Francisco Casacuberta. 2010. Log-linear weight optimisation via Bayesian adaptation in statistical machine translation. In *Proceedings of COLING: Posters*, pages 1077–1085, Beijing.

Sankaran, Baskaran, Anoop Sarkar, and Kevin Duh. 2013. Multi-metric optimization using ensemble tuning. In *Proceedings of HLT/NAACL*, pages 947–957, Atlanta, GA.

Servan, Christophe and Holger Schwenk. 2011. Optimising multiple metrics with MERT. *Prague Bulletin of Mathematical Linguistics*, 96(1):109–117.

Setiawan, Hendra and Bowen Zhou. 2013. Discriminative training of 150 million translation parameters and its application to pruning. In *Proceedings of HLT/NAACL*, pages 335–341, Atlanta, GA.

Shen, Libin, Anoop Sarkar, and Franz Josef Och. 2004. Discriminative reranking for machine translation. In *Proceedings of HLT/NAACL*, pages 177–184, Boston, MA.

Simianer, Patrick, Stefan Riezler, and Chris Dyer. 2012. Joint feature selection in distributed stochastic learning for large-scale discriminative training in SMT. In *Proceedings of ACL*, pages 11–21, Jeju Island.

Sindhwani, Vikas, S. Sathiya Keerthi, and Olivier Chapelle. 2006. Deterministic annealing for semi-supervised kernel machines. In *Proceedings of ICML*, pages 841–848, Pittsburgh, PA.

Smith, David A. and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of COLING/ACL: Poster Sessions*, pages 787–794, Sydney.

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

Sokolov, Artem, Guillaume Wisniewski, and Francois Yvon. 2012a. Computing lattice BLEU oracle scores for machine translation. In *Proceedings of EACL*, pages 120–129, Avignon.

Sokolov, Artem, Guillaume Wisniewski, and François Yvon. 2012b. Non-linear n-best list reranking with few features. In *Proceedings of AMTA*, San Diego, CA.

Sokolov, Artem and François Yvon. 2011. Minimum error rate training semiring. In *Proceedings of EAMT*, pages 241–248, Leuven.

Suzuki, Jun, Kevin Duh, and Masaaki Nagata. 2011. Distributed minimum error rate training of SMT using particle swarm optimization. In *Proceedings of IJCNLP*, pages 649–657, Chiang Mai.

Tan, Ming, Tian Xia, Shaojun Wang, and Bowen Zhou. 2013. A corpus level MIRA tuning strategy for machine translation. In *Proceedings of EMNLP*, pages 851–856, Seattle, WA.

Taskar, Ben, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In *Proceedings of ICML*, pages 896–903, Bonn.

Tibshirani, Robert. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288.

Tillman, Christoph. 2004. A unigram orientation model for statistical machine translation. In *Proceedings of HLT/NAACL*, pages 101–104, Boston, MA.

Tillmann, Christoph and Tong Zhang. 2006. A discriminative global training algorithm for statistical MT. In *Proceedings of COLING/ACL*, pages 721–728, Sydney.

Toutanova, Kristina and Byung-Gyu Ahn. 2013. Learning non-linear features for machine translation using gradient boosting machines. In *Proceedings of ACL: Short Papers*, pages 406–411, Sofia.

Tromble, Roy, Shankar Kumar, Franz Och, and Wolfgang Macherey. 2008. Lattice minimum Bayes-risk decoding for statistical machine translation. In *Proceedings of EMNLP*, pages 620–629, Honolulu, HI.

Turian, Joseph, Benjamin Wellington, and I. Dan Melamed. 2006. Scalable discriminative learning for natural language parsing and translation. In *Proceedings of NIPS*, pages 1409–1416, Vancouver.

Ueffing, Nicola, Gholamreza Haffari, and Anoop Sarkar. 2007. Transductive learning for statistical machine translation. In *Proceedings of ACL*, pages 25–32, Prague.

Ueffing, Nicola, Franz Josef Och, and Hermann Ney. 2002. Generation of word graphs in statistical machine translation. In *Proceedings of EMNLP*, pages 156–163, Philadelphia, PA.

Venugopal, Ashish and Stephan Vogel. 2005. Considerations in maximum mutual information and minimum classification error training for statistical machine translation. In *Proceedings of EAMT*, pages 271–279, Budapest.

Wang, Zhuoran, John Shawe-Taylor, and Sandor Szedmak. 2007. Kernel regression based machine translation. In *Proceedings of HLT/NAACL: Short Papers*, pages 185–188, Rochester, NY.

Watanabe, Taro. 2012. Optimized online rank learning for machine translation. In *Proceedings of HLT/NAACL*, pages 253–262, Montréal.

Watanabe, Taro, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proceedings of EMNLP/CoNLL*, pages 764–773, Prague.

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

Wuebker, Joern, Arne Mauser, and Hermann Ney. 2010. Training phrase translation models with leaving-one-out. In *Proceedings of ACL*, pages 475–484, Uppsala.

Wuebker, Joern, Stephan Peitz, Andreas Guta, and Hermann Ney. 2014. The RWTH Aachen machine translation systems for IWSLT 2014. In *Proceedings of IWSLT*, pages 150–154, Lake Tahoe, NV.

Xiang, Bing and Abraham Ittycheriah. 2011. Discriminative feature-tied mixture modeling for statistical machine translation. In *Proceedings of ACL/HLT*, pages 424–428, Portland, OR.

Xiao, Lin. 2010. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596.

Xiao, Xinyan, Yang Liu, Qun Liu, and Shouxun Lin. 2011. Fast generation of translation forest for large-scale SMT discriminative training. In *Proceedings of EMNLP*, pages 880–888, Edinburgh.

Xiao, Xinyan and Deyi Xiong. 2013. Max-margin synchronous grammar induction for machine translation. In *Proceedings of EMNLP*, pages 255–264, Seattle, WA.

Yamada, Kenji and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of ACL*, pages 523–530, Toulouse.

Yu, Heng, Liang Huang, Haitao Mi, and Kai Zhao. 2013. Max-violation perceptron and forced decoding for scalable MT training. In *Proceedings of EMNLP*, pages 1112–1123, Seattle, WA.

Zaidan, Omar F. and Chris Callison-Burch. 2009. Feasibility of human-in-the-loop minimum error rate training. In *Proceedings of EMNLP*, pages 52–61, Singapore.

Zens, Richard, Sasa Hasan, and Hermann Ney. 2007. A systematic comparison of training criteria for statistical machine translation. In *Proceedings of EMNLP/CoNLL*, pages 524–532, Prague.

Zhao, Bing and Shengyuan Chen. 2009. A simplex Armijo downhill algorithm for optimizing statistical machine translation decoding parameters. In *Proceedings of HLT/NAACL: Short Papers*, pages 21–24, Boulder, CO.

Zhao, Yinggong, Shujie Liu, Yangsheng Ji, Jiajun Chen, and Guodong Zhou. 2011. Transductive minimum error rate training for statistical machine translation. In *Proceedings of IJCNLP*, pages 641–648, Chiang Mai.

Zhou, Liang, Chin-Yew Lin, and Eduard Hovy. 2006. Re-evaluating machine translation results with paraphrase support. In *Proceedings of EMNLP*, pages 77–84, Sydney.

Zinkevich, Martin, John Langford, and Alex J. Smola. 2009. Slow learners are fast. In *Proceedings of NIPS*, pages 2331–2339, Vancouver.