Direct Error Rate Minimization for Statistical Machine Translation

Tagyoung Chung[*]
University of Rochester
Rochester, NY 14627, USA
chung@cs.rochester.edu

Michel Galley
Microsoft Research
Redmond, WA 98052, USA
mgalley@microsoft.com

Abstract

Minimum error rate training is often the preferred method for optimizing parameters of statistical machine translation systems. MERT minimizes error rate by using a surrogate representation of the search space, such as N-best lists or hypergraphs, which only offer an incomplete view of the search space. In our work, we instead minimize error rate directly by integrating the decoder into the minimizer. This approach yields two benefits. First, the function being optimized is the true error rate. Second, it lets us optimize parameters of translations systems other than standard linear model features, such as distortion limit. Since integrating the decoder into the minimizer is often too slow to be practical, we also exploit statistical significance tests to accelerate the search by quickly discarding unpromising models. Experiments with a phrase-based system show that our approach is scalable, and that optimizing the parameters that MERT cannot handle brings improvements to translation results.

1 Introduction

Minimum error rate training (Och, 2003) is a common method for optimizing linear model parameters, which is an important part of building good machine translation systems. MERT minimizes an arbitrary loss function, usually an evaluation metric such as BLEU (Papineni et al., 2002) or TER (Snover et al., 2006) from a surrogate representation of the search space, such as the N-best candidate translations of a development set. Much of the recent work on minimum error rate training focused on improving the method by Och (2003). Recent efforts extended MERT to work on lattices (Macherey et al., 2008) and hypergraphs (Kumar et al., 2009). Random restarts and random walks (Moore and Quirk, 2008) are commonly used to combat the fact the search space is highly non-convex, often with multiple minima.

Several problems still remain with MERT, three of which are addressed by this work. First, the N-best error surface explored by MERT is generally not the same as the true error surface, which means that the error rate at an optimum[1] of the N-best error surface is not guaranteed to be any close to an optimum of the true error surface. Second, most SMT decoders make search errors, yet MERT ignores the fact that the error surface of an error-prone decoder differs from the one of an exact decoder (Chang and Collins, 2011). MERT calculates an envelope from candidate translations and assumes all translations on the envelope are reachable by the decoder, but these translations may become unreachable due to search errors. Third, MERT is only used to tune linear model parameters, yet SMT systems have many free decoder parameters—such as distortion limit and beam size—that are not handled by MERT. MERT does not provide a principled way to set these parameters.

In order to overcome these issues, we explore the application of direct search methods (Wright, 1995) to SMT. To do this, we integrate the decoder and the evaluation metric inside the objective function,

[*] This research was conducted during the author's internship at Microsoft Research.

[1]The optimum found by MERT (Och, 2003) is generally not globally optimal. An alternative that optimizes N-best lists exactly is presented by Galley and Quirk (2011), and we do not discuss it further here.

which takes source sentences and a set of weights as inputs, and outputs the evaluation score (e.g., BLEU score) computed on the decoded sentences. Since it is impractical to calculate derivatives of this function, we use derivative-free optimization methods such as the downhill simplex method (Nelder and Mead, 1965) and Powell's method (Powell, 1964), which generally handle such difficult search conditions relatively well. This approach confers several benefits over MERT. First, the function being optimized is the true error rate. Second, integrating the decoder inside the objective function forces the optimizer to account for possible search errors. Third, contrary to MERT, our approach does not require input parameters to be those of a linear model, so our approach can tune a broader range of features, including non-linear and hidden-state parameters (e.g., distortion limit, beam size, and weight vector applied to future cost estimates).

In this paper, we make direct search reasonably fast thanks to two speedup techniques. First, we use a model selection acceleration technique called *racing* (Moore and Lee, 1994) in conjunction with randomization tests (Riezler and Maxwell, 2005) to avoid decoding the entire development set at each function evaluation. This approach discards the current model whenever performance on the translated subset of the development data is deemed significantly worse in comparison to the current best model. Second, we store and re-use search graphs across function evaluations, which eliminates some of the redundancy of regenerating the same translations in different optimization steps.

Our experiments with a strong phrase-based translation system show that the direct search approach is an effective alternative to MERT. The speed of direct search is generally comparable to MERT, and translation accuracy is generally superior. The non-linear and hidden-state features tuned in this work bring gains on three language pairs, with improvements ranging between 0.27 and 0.35 BLEU points.

2 Direct error rate minimization

Most current machine translation systems use a log-linear model:

$$p(e|f) \ \propto \ \exp\Big(\sum_i \lambda_i h_i(e,f)\Big)$$

where f is a source sentence, e is a target sentence, h_i is a feature function, and λ_i is the weight of this feature. Given a source sentence f, finding the best target sentence $\hat{e}$ according to the model is a search problem, which is called decoding:

$$\hat{e} = \operatorname*{argmax}_e \ \exp\Big(\sum_i \lambda_i h_i(e,f)\Big)$$

The target sentence $\hat{e}$ is automatically evaluated against a reference translation r using any metric that is known to be relatively well correlated with human judgment, such as BLEU or TER. Let us refer to such error function as $\mathrm{E}(\cdot)$. Then, the process of finding the best set of weights $\hat{\boldsymbol{\lambda}}$ according to an error function E is another search:

$$\hat{\boldsymbol{\lambda}} = \operatorname*{argmin}_{\boldsymbol{\lambda}} \mathrm{E}\Bigg(r; \operatorname*{argmax}_e \ \exp\Big(\sum_i \lambda_i h_i(e,f)\Big)\Bigg)$$

The typical MERT process solves the problem in an iterative fashion. At each step i, it produces N-best lists by decoding with $\hat{\boldsymbol{\lambda}}_i$, then uses these lists to find $\hat{\boldsymbol{\lambda}}_{i+1}$. Och (2003) presents an efficient multi-directional line search algorithm, which is based on the fact that the error count along each line is piecewise constant and thus easy to optimize exactly. The process is repeated until a certain convergence criterion is met, or until no new candidate sentences are added to the pool. The left side of Figure 1 summarizes this process.

Though simple and effective, there are several limitations to this approach. The primary reason is that it can only tune parameters that are part of the log-linear model. Aside from having parameters from the log-linear model, decoders generally have free parameters $\boldsymbol{\theta}$ that needs to be set manually, such as beam size and distortion limit. These decoder-related parameters have complex interactions with linear model parameters, thus, ideally, we would want to tune them jointly with decoder parameters such as distortion limit.

Direct search addresses these problems by including all feature parameters and all decoder-related parameters within the optimization framework. Figure 1 contrasts MERT with direct search. Rather than optimizing candidate pools of translations, direct search treats the decoder and the evaluation tool

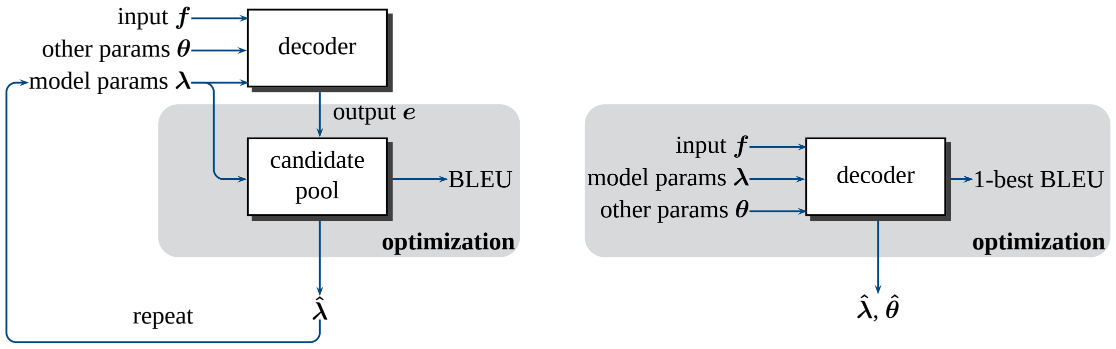
input $\boldsymbol{f}$
other params $\boldsymbol{\theta}$
model params $\boldsymbol{\lambda}$
decoder
output $\boldsymbol{e}$
candidate pool
BLEU
optimization
$\hat{\boldsymbol{\lambda}}$
repeat
input $\boldsymbol{f}$
model params $\boldsymbol{\lambda}$
other params $\boldsymbol{\theta}$
decoder
1-best BLEU
optimization
$\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\theta}}$

Figure 1: Comparison of MERT (left) and direct search (right).

as a single function:

$$\Phi(f, r; \boldsymbol{\lambda}, \boldsymbol{\theta}) = \mathrm{E}\Big(r; \underset{e}{\operatorname{argmax}} \ \exp\big(\sum_i \lambda_i h_i(e, f)\big)\Big)$$

Then, it uses an optimization method to minimize the function:

$$\underset{\boldsymbol{\lambda},\boldsymbol{\theta}}{\operatorname{argmin}} \ \Phi(f, r; \boldsymbol{\lambda}, \boldsymbol{\theta})$$

This formulation solves the problem mentioned previously, since we jointly optimize $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$, thus accounting for the dependencies between the two. However, there are two problems to address with direct error minimization. First, this approach requires the entire development set to be re-decoded every time the function is evaluated, which can be prohibitively expensive. To address this problem, we present several methods to speed up the search process in Section 5. Second, since the gradient of standard evaluation metrics such as BLEU is not known and since methods for estimating the gradient numerically require too many function evaluations, we cannot use common search methods that use derivatives of a function. Therefore, we need robust derivative-free optimization methods. We discuss such optimization methods in Section 3.

3 Derivative-free optimization

As discussed in the previous sections, we need to rely on derivative-free optimization methods for direct search. We consider two such optimization methods:

Powell's method For each iteration, Powell's method tries to find a good direction along which the function can be minimized. This direction is determined by searching along each standard base vector. Then, a line search is performed along the direction by using line search methods such as golden section search or Fibonacci search. The process is repeated until convergence. We implement the golden section search as presented by Press et al. (1992) in our experiments. Although the golden section search is only exact when the function is unimodal, we found that it works quite well in practice. More details are presented by Powell (1964).

Nelder-Mead method This approach sets up a simplex on the search space, which is a polytope with $D + 1$ vertices when there are D dimensions, and successively moves the simplex to a lower point to find a minimum of the function. The simplex is moved using different actions, which are taken when certain conditions are met. The basic idea behind these actions is to replace the worst point in the simplex with a new and better point, thereby moving the simplex towards a minimum. This method has the advantage of being able to deal with "bumpy" functions and depending on the configuration of the simplex at the time, it is possible to escape some local minima. This is often refer to as downhill simplex method and more details are presented by Nelder and Mead (1965).

4 Parameters

In this section, we discuss the parameters that we optimize with direct search, in addition to standard linear model parameters:

4.1 Distortion limit

Distortion limit is one of decoder parameters that sets a limit on the number of words the decoder is allowed to skip when deciding which source phrase to translate in order to allow reordering. Figure 2 shows a translation example from English to Japanese. Every word jumped over incurs a distortion cost, which is usually one of the translation model parameters, which thereby discourages reordering of words unless language model supports the reordering.

Since having a large distortion limit leads to slower decoding, having the smallest possible distortion limit that still facilitates correct reordering would be ideal. Not only this speeds up translation, but this also leads to better translation quality by minimizing search errors. Since a larger distortion limit means there are more possible re-orderings of translations, it is prone to more search errors. In fact, there are evidences that tuning the distortion limit is beneficial in improving quality of translation by limiting search errors. Galley and Manning (2008) conduct a line search along increments of distortion limit and separately tune the translation model parameters for each increment of distortion limit. The result shows significant difference in translation quality when distortion limit is tuned along with the model parameters. Separately tuning model parameters for different distortion limit is necessary because model parameters are coupled with distortion limit. A representative example: when distortion limit is zero, the distortion penalty feature can have any weight and not affect BLEU scores, but this is not the case when distortion limit is larger than zero. Tuning distortion limit in direct search in conjunction with related features such linear distortion eliminates the need for a line search for distortion limit.

4.2 Polynomial features

Most phrase-based decoders typically use a distortion penalty feature to discourage (or maybe sometimes encourage) reordering. Whereas distortion limit is a hard constraint—since the decoder never considers jumps larger than the given limit—distortion penalty is a soft constraint, since it penalizes reordering proportionally to the length of the jump. The total distortion penalty is calculated as follows:

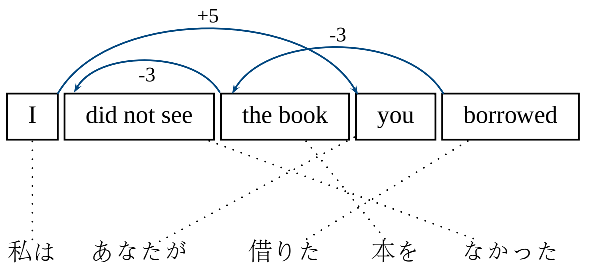

Figure 2: Reordering in phrase-based translation. A minimum distortion limit of five is needed to correctly translate this example. The source sentence is relatively simple but a relatively large distortion limit is needed to accommodate the correct reordering due to typological difference between two languages.

$$D(e, f) = \lambda_d \sum_j |d_j|^{p_d}$$

where λ_d is the weight for distortion penalty feature, and d_j is the size of the jump needed to translate the j-th phrase pair. For example, in Figure 2, the total distortion penalty feature value is 11, which is multiplied with λ_d to get the total distortion cost of translating the example sentence. Although p_d is typically set to one (linear), one may consider polynomial distortion penalty (Green et al., 2010). Green et al. (2010) show that setting p_d to a higher value than one improves the translation quality, but uses a predetermined value for p_d. Instead of manually setting the value of p_d, it can be given a value tuned with direct search. Although we only discussed distortion penalty here, it is straightforward to tune p_i for each feature $h_i(e, f)^{p_i}$ using direct error rate minimization, where $h_i(e, f)$ is any linear model feature of the decoder.

4.3 Future cost estimates

Since beam search involves pruning, it is crucial to have good future cost estimation in order to minimize the number of search errors (Koehn et al., 2003). The concept of future cost estimation is related to heuristic functions in the A* search algorithm. The total cost $f(x)$ of a partial translation hypothesis is estimated by combining $g(x)$, which is the actual current cost from the beginning of a sentence to point x and $h(x)$, which is the future cost estimate from point x to the end of the sentence:

$$f(x) = g(x) + h(x)$$

In SMT decoding, the same feature weight vector is generally used when computing $g(x)$ and $h(x)$. However, this may not be ideal since future cost estimators use different heuristics depending on the features. For example, the future cost estimator (Green et al., 2010) for linear distortion always underestimates completion cost, which is generally deemed a good property. Unfortunately, some features have estimators that tend to overestimate completion cost, as it is the case with the language model. This problem is illustrated in Figure 3. The Figure shows that the ratio between the estimated total cost and the actual total cost converges to 1.0. However, in earlier stages of translations, the estimated future cost for language model is larger than it should be, which leads to higher total estimated cost. In the A* search parlance, we are using an inadmissible heuristic since the future cost is overestimated, which leads to suboptimal search. This suggests that separately tuning parameters that are involved in the future cost estimation will lead to better pruning decisions. This essentially doubles the number of linear model parameters, since for every feature used in future cost estimation, we create a counterpart and tune its weight independently.

4.4 Search parameters

In addition to the parameters listed above, we also tune general decoder parameters that affect the search quality: beam size and parameters controlling histogram pruning and threshold pruning. While it makes sense to set these parameters automatically instead of manually, the methods we have presented thus far are not particularly fit for this type of parameters. Indeed, if the sole goal is to maximize translation quality (e.g., as measured by standard BLEU), a larger beam size and less pruning is usually preferable. To address this problem, we optimize these three parameters using a slightly different objective function. When tuning any of these three features, the goal of translation is to get the most accurate translation given a pre-defined time limit, so we change the objective to be a time-sensitive objective function. Much akin to brevity penalty in BLEU,

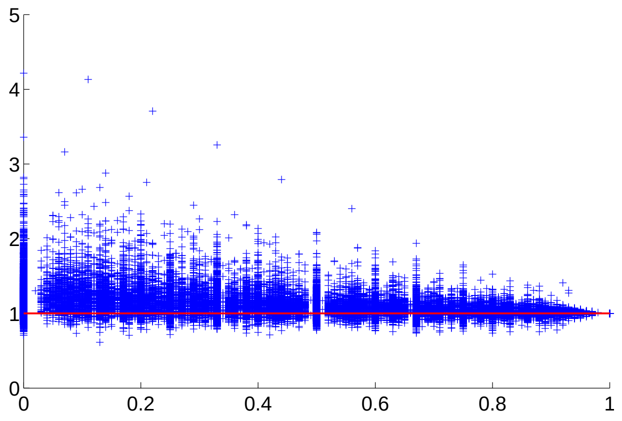

Figure 3: y axis is ratio between estimated total cost vs. actual total cost of language model for thousands of translations. 1.0 means the estimated total cost and the actual total cost are exactly the same, and anything higher than 1.0 means the future cost has been overestimated thereby inflating the estimated total cost. The x-axis represents how much translation has been completed. 0.1 means 10% of a sentence has been translated.

we define time penalty as:

$$\mathbf{TP}(\,\cdot\,) = \begin{cases} 1.0 & t_i \leq t_d \\ \exp\left(1 - \frac{t_i}{t_d}\right) & t_i > t_d \end{cases}$$

where $\mathbf{TP}(\,\cdot\,)$ is a time penalty that is multiplied to BLEU, t_i is the time it takes to translate development set under current parameters, and t_d is the desired time limit for translating the development set. With this error metric, we still optimize for the translation quality as long as the translation happens within desired time t_d. With the modified time-sensitive BLEU score as error metric, direct search may tune the parameters that have the speed and accuracy trade-off that we want.[2]

5 Speeding up direct search

Optimizing the true error surface is generally more computationally expensive than with any surrogate error surface, since each function evaluation usually requires decoding or re-decoding the entire development set. Since SMT tuning sets used for error

[2]A disadvantage of using time in the definition of $\mathbf{TP}(\,\cdot\,)$ is that it adds non-determinism that can make optimization unstable. Our solution is to replace time with pseudo-time, a deterministic substitute expressed as a linear combination of the number of n-gram lookups and hypothesis expansions (these two quantities correlate quite well with decoding time).

rate minimization often comprise one thousand sentences or more, each function evaluation can take minutes or more. However, this problem is somewhat mitigated by the fact that translating in batches is highly parallelizable. Since MERT (Och, 2003) is also easily parallelizable, we need to resort to other speedup techniques to make direct search a practical alternative to MERT. We now present two techniques that make optimization of the true error surface more efficient.

5.1 A racing algorithm for speeding up SMT model selection

Error rate minimization as presented in this paper can be seen as a form of model selection, which has been the focus of a lot of work in the learning literature. The most popular approaches to model selection—such as minimizing cross validation error—tend to be very slow in practice; therefore, researchers have addressed the problem of accelerating model selection using statistical tests.

Prior to considering the SMT case, we review one of these methods in the case of leave-one-out cross validation (LOOCV). *Racing* for model selection (Maron and Moore, 1994; Moore and Lee, 1994) works as follows: we are given a collection of N_m models and N_d data points, and we must find the model that minimizes the mean $e^*_j = \frac{1}{N_d}\sum_i e_j(i)$, where $e_j(i)$ is the classification error of model M_j on the ith datapoint when trained on all datapoints except the ith point. The models are evaluated concurrently, and at any given step $k \in [1, N_d]$, each model M_j is associated with two pieces of information: the current estimate of its mean error rate, and the estimate of its variance. As evaluations progress, we eliminate any model that is significantly worse than any other model.[3] We also note that the Racing technique first randomizes the order of the data points to ensure that prefixes of the dataset are generally representative of the entire set.

In this work, we use Racing to speed up direct search for SMT, but this requires two main adjustments compared to the LOOCV case. First, our models have real-valued parameters, so we cannot exhaustively evaluate the set of all models since it is infinite. Instead, we use direct search to select which models compete against each other during Racing. In the case of Powell's method, all points of a grid along the current search direction are evaluated in parallel using Racing, before we turn to the next line search. In the case of the downhill simplex optimizer and in the case of line searches other than grid search (e.g., golden section search), the use of Racing is more difficult because the function evaluations requested by these optimizers have dependencies that generally prevent concurrent function evaluations. Since functions in downhill simplex are evaluated in sequence and not in parallel, our solution is to race the current model against our current best model.[4] When the evaluation of a model M is interrupted because it is deemed significantly worse than the current best model $\hat{M}$, the error rate of M on the entire development set is extrapolated from its relative performance on the decoded subset.[5]

The second main difference with the LOOCV case is that we do not use confidence intervals to determine which of two or models are best. In SMT, it is common to use either bootstrap resampling (Efron and Tibshirani, 1993; Och, 2003) or randomization tests (Noreen, 1989). In this paper, we use the randomization test for discarding unpromising models, since this statistical test was shown to be less likely to cause type-I errors[6] than bootstrap methods (Riezler and Maxwell, 2005). Since both kinds of statistical tests involve a time-consuming sampling step, it

[3]The details of these statistical tests are not so important here since we use different ones in the case of SMT, but we briefly summarize them as follows: Maron and Moore (1994) use a non-parametric method (Hoeffding bounds (Hoeffding, 1963)) for confidence estimation, and places confidence intervals on the mean value of the random variable representing $e_j(i)$. A model is discarded if its confidence interval no longer overlaps with the confidence interval of the current best model. Moore and Lee (1994) use a similar technique, but relies on Bayesian statistics instead of Hoeffding bounds.

[4]Since Racing only discards suboptimal models, the current best model M^* is one for which we have decoded the entire development set. Once a new model M is evaluated, we perform at step j a significance test to determine whether M's translation of sentences $1 \dots j$ is better or worse than M^* translation for the same range of sentences. If M is significantly worse, we discard it. If M^* is worse, we continue evaluating the performance of M, since we need M's output for the full development set if M eventually becomes the new best model.

[5]For example, if error rates of $\hat{M}$ and M are respectively 10% and 11% on the subset decoded by both models and $\hat{M}$'s error on the entire set is 20%, M's extrapolated error is 22%.

[6]A type I error rejects a null hypothesis that is true.

is somewhat wasteful to perform a new test after the decoding of each sentence, so we translate sentences in small batches of K sentences before performing each randomization test.[7]

We finally note that Racing no longer guarantees that the error function observed by the optimizer is the true error function. Racing causes some approximations of the error function, but the degree of approximation is designed to be small in regions with low error rates, and Racing ensures that the most promising function evaluations in our progression towards an optimum are unaffected. In contrast, the approximation of the error function computed from N-best lists or lattice does not share this property.[8]

To further speed up function evaluations in direct search, we employ a method meant to deal with models that are nearly identical, a situation in which Racing usually does not help much. Indeed, when two models produce very similar outputs, we often need to run the race through every sentence of the development set since none of the two models end up being significantly better. A solution to this problem consists of discarding models that are nearly identical to other models, where similarity between models is solely measured from their outputs.[9] To do this, we resort again to a randomization test: Given two models M_a and M_b, this test performs random permutations between outputs of M_a and M_b, that is, it determines for each sentence of index i whether or not to permute the two model outputs, with probability $p = 0.5$. When M_a and M_b are very similar, these permutations have little effect, even when we repeat this sampling process many times. To cope with this problem, we slightly modify the randomization test to discard one of the two nearly identical models. Specifically, we compute the gap—measured in error rate—between the best randomized output and the worst randomized output. If this gap is lower than a pre-defined threshold, we only keep the best model.[10] This adjustment to the significance test makes direct search reasonably fast, since Racing is effective during the initial steps of search (when steps tend to be relatively big, and when differences in error rate are pretty significant), and our modification to randomization tests helps while search converges towards an optimum using increasingly smaller steps.

[7]In our experiments, we set $K = 50$. Some other practical considerations: the significance level used for discarding unpromising models is $p \leq .05$. The randomization test is a sampling-based technique, for which we must specify a sample size R. In this paper, we use $R = 5000$.

[8]In the case of N-best MERT, it is not even guaranteed that we find the true error rate of our current best model M while searching the N-best error surface. In fact, if we take the parameters of our best model M and re-decode the development set, we may get an error rate that is different from what was predicted from the N-best list. With direct search and Racing, no such approximation affects our current best model.

[9]Measuring model similarity only based on parameter values is less effective, since features and other parameters are sometimes redundant, and two models may behave similarly while having fairly distinct parameter values.

5.2 Lattice-based decoding

We use another technique to speed up direct search by storing and re-using search graphs, which consist of lattices in the case of phrase-based decoding (Och et al., 1999) and hypergraphs in the case of hierarchical decoding (Chiang, 2005). The successive expansion of translation options in order to construct the search graph is generally done from scratch, but this can be wasteful when the same sentences are translated multiple times, as it is the case with direct search. Even when the parameters of the decoder change across function evaluations, some partial translation are likely to be constructed multiple times, and this is more likely to happen when changes in parameters are relatively small. To overcome this inefficiency, we memoize hypotheses expansions made in all function evaluations, which then allows us to reuse some edges (or hyperedges) from previous iterations to construct the current graph (or hypergraph). Since feature values—including expensive features like language model score—are stored into each edge, the speedup is roughly proportional to the percentage of edges we can reuse.

A more radical way of exploiting search graphs of previous iterations is to use them as constraints in a forced decoding approach. In this framework, the decoder takes as input not only an input sentence, but also a constraining search graph. During decoding, it is forced to discard any translation hypothe-

[10]In the case where we compare our current best model and a model that is currently being evaluated, we discard the latter. In our experiments with BLEU, we discard if the gap is smaller than 0.1 BLEU point.

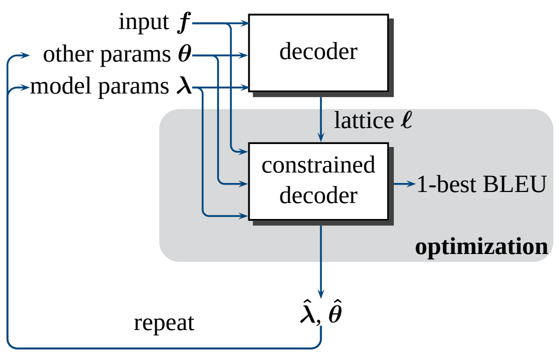
input $\boldsymbol{f}$
other params $\boldsymbol{\theta}$
model params $\boldsymbol{\lambda}$
decoder
lattice $\boldsymbol{\ell}$
constrained decoder
1-best BLEU
optimization
repeat
$\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\theta}}$

Figure 4: Lattice-constrained decoding for direct search.

ses that violate the constraining search graph. This makes the memoization method presented in the previous paragraph maximally efficient, since lattice-constrained decoding has all linear model feature values already pre-computed. While this approach is similar in spirit to lattice-based MERT (Macherey et al., 2008), there is a crucial difference. The optimization steps in lattice MERT bypass the decoder, but the lattice-based approach presented here does not. The distinction is important when it comes to tuning non-linear and hidden state parameters of the decoder. For instance, the initial lattice may have been constructed with a distortion limit of 4, while the current model specifies a distortion limit of 2. At that stage, optimization via lattice-constrained decoding instead of lattice-based MERT ensures that we will never select a path of the input lattice that corresponds to a distortion limit of more than 2. This is important since the error rate must reflect the fact that jumps of two or more words are not allowed.

Figure 4 shows how direct search with lattice-constrained decoding is structured. Similarly to MERT and as opposed to straight direct search, optimization is repeated multiple times. Since each optimization in the lattice-constrained case does not require recomputing any features, it usually turns into very significant gains in terms of translation speed, though it also causes a small loss of translation accuracy in general. The overall approach depicted in Figure 4 works as follows: a first set of lattices is generated using an initial $\boldsymbol{\lambda}_0$ and $\boldsymbol{\theta}_0$. We then run direct search with a decoder constrained on this set of lattices. After optimization has converged, the optimal $\hat{\boldsymbol{\lambda}}$ and $\hat{\boldsymbol{\theta}}$ are provided as input $\boldsymbol{\lambda}_1$ and $\boldsymbol{\theta}_1$ to start a new iteration of this process. Note that the constraining lattices built at each iteration are always merged with those of the previous ones, so constraining lattices grow over time. The two stopping criteria are similar to MERT: if the norm of the difference between the previous parameter vector—including $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$—and the current vector falls below a predefined tolerance value, we do not continue to the next iteration. Alternatively, if a new pass of unconstrained decoding generates lattices that are subsumed by lattices constructed at previous iteration, we stop and do not run the next optimization step.

	Train	MERT dev.	Test
Korean-English	7.9M	1000	6000
Arabic-English	11.1M	1000	6000
Farsi-English	739K	1000	2000

Table 1: Size of bitexts in number of sentence pairs.

6 Experiments

6.1 Setup

For our experiments, we use a phrase-based translation system similar to Moses (Koehn et al., 2007). Our decoder uses many of the same features as Moses, including four phrasal and lexicalized translation scores, phrase penalty, word penalty, a language model score, linear distortion, and six lexicalized reordering scores. Unless specified otherwise, the decoder's stack size is 50, and the number of translation options per input phrase is 25.

Table 1 summarizes the amount of training data used to train translation systems from Korean, Arabic, and Farsi into English. These data sets are drawn from various sources, which include news, web, and technical data, as well as United Nations data in the case of Arabic. In order to get the sense of how presented techniques generalize, we evaluate our systems on a fairly broad domain. We use development and test sets are a mix of news, web, and technical data. All systems translate into English, for which we built a 5-gram language model with cutoff counts 1, 1, 1, 2, 3 for unigrams to 5-grams, using a corpus of roughly seven billion English words. This includes the target side of the parallel training data, plus a significant amount of data gathered from the web.

#	Minimizer	Optimized parameters	Arabic		Korean		Farsi	
1	MERT with grid search	lin, DL	29.12	(14.6)	23.30	(20.8)	32.16	(11.7)
2	Direct search (simplex)	lin, DL	29.07	(1.2)	23.42	(4.4)	32.22	(1.3)
3	Direct search (Powell)	lin, DL	29.20	(2.3)	23.39	(5.6)	32.28	(2.1)
4	Direct search (Powell)	lin, extended, DL	29.39	(4.4)	23.61	(8.9)	32.51	(4.9)
5	Lattice-constrained (Powell)	lin, extended, DL	29.27	(0.7)	23.43	(1.3)	32.42	(1.1)
6	Direct search (Powell)	lin, extended, DL, search	29.31	(6.5)	23.46	(9.7)	32.62	(6.2)

Table 2: BLEU-4 scores (%) with one reference, translating into English; the numbers in parentheses are times in hours to run parameter optimization end-to-end. 'Lin' refers to Moses linear model features; 'extended' refers to non-linear and hidden state features (polynomial features, future cost); 'DL' refers to distortion limit; 'search' is the set of parameters controlling search quality (parameters controlling beam size, histogram pruning, and threshold pruning).

Our baseline system is trained for each language pair by running minimum error rate training (Och, 2003) on 1000 sentences. Each iteration of MERT utilizes 19 random starting points, plus the points of convergence at all previous iterations of MERT, and a uniform weight vector. That is, the first iteration of MERT uses 20 starting points, the second uses 21 points, etc. Since MERT is not able to directly optimize search parameters such as distortion limit and beam size, our baseline system uses grid search to optimize them. To make this search more tractable, we only perform the grid search for a single parameter: the distortion limit. For each language pair, the grid search consists of repeating MERT for eight distinct distortion limits ranging from 3 to 10. The optimal distortion limits found for Korean, Arabic, and Farsi, are 8, 5, and 6, respectively.[11] To ensure that the comparison with our approach is consistent, this grid search is made on the MERT dev set itself.

The next subsection contrasts the different direct search methods presented in this paper. Note that all these experiments use the speedup techniques based on statistical significance test presented in Section 5. Indeed, we found that using these techniques resulted in faster speeds without affecting the search in any significant way. Models tuned with or without significance tests often ended up identical.

6.2 Results

The main results are shown in Table 2, and are computed using standard BLEU-4 (Papineni et al., 2002) using one reference translation, and ignoring case. Row 1 displays results of the MERT baseline, with a distortion limit that was found optimal using a grid search on the development set. Rows 2 and 3 show results of direct error rate minimization with downhill simplex and Powell's method, where direct search optimizes both linear model parameters and the distortion limit. We see here that the performance of direct search is comparable and sometimes better than MERT, but the benefit of direct search here is that it does not require an external grid search to find an effective distortion limit (each direct search is initialized with a distortion limit of 10). Row 4 shows the performance of Powell's method using the extended parameter set (Section 4), which includes model weights for future costs and polynomial features. We lack space to present an extensive analysis of the relative impact of the different non-linear features and parameters discussed in this paper, but we generally find that the following parameters work best: distortion limit, polynomial distortion penalty, and weight of future cost estimate of the language model. The fact that Moses-style future cost estimation for language models often overestimates probably explains why the latter feature helps.

In the last row of Table 2, optimization is done using the time-sensitive variant of BLEU presented in Section 4.4, and the set of parameters tuned here includes all the previous ones, in addition to beam size, and the two parameters controlling histogram and threshold pruning in beam search. Clearly, running direct search to directly optimize BLEU would yield a very large beam size and would set pruning parameters that are so permissive that they would almost completely disable pruning. The benefit of using the time-sensitive variant of BLEU is that direct

[11] We rerun MERT for each different distortion limit because of the dependencies between this parameter and linear model features, particularly linear distortion and lexicalized reordering scores. A linear model that is effective with a distortion limit of 4 can be suboptimal for a limit of 8.

search is forced to find parameter weights that offer a good balance between accuracy and speed. To make our results in row 6 as comparable as possible to row 4, we use the running time (on the development set) of row 4 as a time constraint for the model of row 6, which is to decode the entire development set at least as fast. In other words, the system of row 6 is optimized to be no slower than the system of row 4, and is otherwise penalized due to the time penalty. The effect of this is that translation speed at tuning time is almost the same, and speed of systems 4 and 6 is roughly the same at test time. A comparison between rows 4 and 6 suggests that tuning search parameters such as beam size and without affecting time does not provide much gain in terms of translation quality, but the method nevertheless has one advantage: one can target a specific translation speed without having to manually tune any parameter such as beam size, and without even having to decide which parameter to manually tune.

Times to run optimizations end-to-end are reported in parentheses in Table 2 and they take into account the time to run the grid search in the case of MERT. Times to decode test sets are not reported here since they are roughly the same across all models. While translation accuracy with MERT and direct search is roughly the same when the underlying parameter set is the same, direct search wins in running time when it comes to optimizing search parameters like distortion limit. Since each grid search runs MERT eight times, MERT is generally faster than direct search, but the difference of speed remains reasonable if the number of tuned parameters is the same, and direct search is rarely twice as slow.

We finally discuss the case of lattice-constrained decoding, which is shown in row 5 of Table 2. This method is not applicable when tuning parameters that affect search thoroughness (row 6), such as beam size. The reason is that lattice-constrained decoding is a form of forced decoding that considerably narrows the search space. Under a constrained decoding setting, it appears that a large beam size seldom affects translation speed, but this is misleading and largely due to constraints created by the lattice. We thus evaluate the lattice-constrained case without tuning 'search' features, and find that direct search is significantly faster using lattice-constrained, with only a slight degradation of translation quality. Lattice constraints are augmented 2-5 times before it converges.

7 Related work

The use of derivative-free optimization methods to tune machine translation parameters has been tried before. Bender et al. (2004) used the Nelder-Mead method to tune model parameters for a phrase-based translation system. However, their way of making direct search fast and practical is to set distortion limit to zero, which results in poor translation quality for many language pairs. Zens et al. (2007) also use the Nelder-Mead method to tune parameters in a log-linear model to maximize expected BLEU. Zhao and Chen (2009) proposes changes to Nelder-Mead method to better fit parameter tuning in their machine translation setting. They show the modification brings better search of parameters over the regular Nelder-Mead method. Our work is related to the search-based structured prediction (SEARN) model of Daumé (2006), in the sense that direct search also accounts for what happens during search (including search errors) to try to find parameters that are not only good for prediction, but for search as well.

8 Conclusion

This paper addressed the problem of minimizing error rate at a corpus level. We show that a technique to directly minimize the true error rate, rather than one estimated from a surrogate representation such as an N-best list, is in fact feasible. We present two techniques that make this minimization significantly faster, to the point where this technique is a viable alternative to MERT. In the case where free parameters of the decoder (such as distortion limit) also need to be optimized, our technique is in fact much faster. We also optimize non-linear and hidden state features that cannot be tuned using MERT, which yield improvements in translation accuracy. Experiments on large test sets yield gains on three language pairs, and our best configuration outperforms MERT by 0.27 to 0.35 BLEU points using a baseline system trained on large amounts of data.

Acknowledgments

We thank anonymous reviewers, Chris Quirk, Kristina Toutanova, and Anthony Aue for valuable suggestions.

References

Oliver Bender, Richard Zens, Evgeny Matusov, and Hermann Ney. 2004. Alignment templates: the RWTH SMT system. In *Proc. of the International Workshop on Spoken Language Translation*, pages 79–84, Kyoto, Japan.

Yin-Wen Chang and Michael Collins. 2011. Exact decoding of phrase-based translation models through Lagrangian relaxation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 26–37, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Conference of the Association for Computational Linguistics (ACL-05)*, pages 263–270, Ann Arbor, MI.

Hal Daumé, III. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California, Los Angeles, CA, USA.

B. Efron and R. J. Tibshirani. 1993. *An Introduction to the Bootstrap*. Chapman & Hall, New York.

Michel Galley and Christopher D. Manning. 2008. A simple and effective hierarchical phrase reordering model. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 848–856, Honolulu, Hawaii, October. Association for Computational Linguistics.

Michel Galley and Chris Quirk. 2011. Optimal search for minimum error rate training. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 38–49, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

Spence Green, Michel Galley, and Christopher D. Manning. 2010. Improved models of distortion cost for statistical machine translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 867–875, Los Angeles, California, June. Association for Computational Linguistics.

Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-03)*, Edmonton, Alberta.

Philipp Koehn, 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, Demonstration Session*, pages 177–180.

Shankar Kumar, 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 the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 163–171, Suntec, Singapore, August. Association for Computational Linguistics.

Wolfgang Macherey, Franz Och, Ignacio Thayer, and Jakob Uszkoreit. 2008. Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 725–734, Honolulu, Hawaii, October. Association for Computational Linguistics.

Oded Maron and Andrew W. Moore. 1994. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in neural information processing systems 6*, pages 59–66. Morgan Kaufmann.

Andrew Moore and Mary Soon Lee. 1994. Efficient algorithms for minimizing cross validation error. In W. W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Confonference on Machine Learning*, pages 190–198. Morgan Kaufmann.

Robert C. Moore and Chris Quirk. 2008. Random restarts in minimum error rate training for statistical machine translation. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 585–592, Manchester, UK, August. Coling 2008 Organizing Committee.

J. A. Nelder and R. Mead. 1965. A simplex method for function minimization. *Computer Journal*, 7:308–313.

Eric W. Noreen. 1989. *Computer-Intensive Methods for Testing Hypotheses : An Introduction*. Wiley-Interscience.

Franz Josef Och, Christoph Tillmann, Hermann Ney, and Lehrstuhl Fiir Informatik. 1999. Improved alignment models for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

Franz Josef Och. 2003. Minimum error rate training for statistical machine translation. In *Proceedings of the 41th Annual Conference of the Association for Computational Linguistics (ACL-03)*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*.

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

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 1992. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA.

Stefan Riezler and John T. Maxwell. 2005. On some pitfalls in automatic evaluation and significance testing for MT. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 57–64, June.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*, pages 223–231.

M H Wright. 1995. Direct search methods: Once scorned, now respectable. *Numerical Analysis*, 344:191–208.

Richard Zens, Sasa Hasan, and Hermann Ney. 2007. A systematic comparison of training criteria for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 524–532.

Bing Zhao and Shengyuan Chen. 2009. A simplex Armijo downhill algorithm for optimizing statistical machine translation decoding parameters. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 21–24, Boulder, Colorado, June. Association for Computational Linguistics.