Large-Scale Discriminative Training for Statistical Machine Translation Using Held-Out Line Search

Jeffrey Flanigan **Chris Dyer** **Jaime Carbonell**
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{jflanigan,cdyer,jgc}@cs.cmu.edu

Abstract

We introduce a new large-scale discriminative learning algorithm for machine translation that is capable of learning parameters in models with extremely sparse features. To ensure their reliable estimation and to prevent overfitting, we use a two-phase learning algorithm. First, the contribution of individual sparse features is estimated using large amounts of parallel data. Second, a small development corpus is used to determine the relative contributions of the sparse features and standard dense features. Not only does this two-phase learning approach prevent overfitting, the second pass optimizes corpus-level BLEU of the Viterbi translation of the decoder. We demonstrate significant improvements using sparse rule indicator features in three different translation tasks. To our knowledge, this is the first large-scale discriminative training algorithm capable of showing improvements over the MERT baseline with only rule indicator features in addition to the standard MERT features.

1 Introduction

This paper is about large scale discriminative training of machine translation systems. Like MERT (Och, 2003), our procedure directly optimizes the cost of the Viterbi output on corpus-level metrics, but does so while scaling to millions of features. The training procedure, which we call the **Held-Out Line Search** algorithm (HOLS), is a two-phase iterative batch optimization procedure consisting of (1) a gradient calculation on a differentiable approximation to the loss on a large amount of parallel training data and (2) a line search (using the standard MERT algorithm) to search in a subspace defined by the gradient for the weights that minimize the true cost.

While sparse features are successfully used in many NLP systems, such parameterizations pose a number of learning challenges. First, since any one feature is likely to occur infrequently, a large amount of training data is necessary to reliably estimate their weights. Therefore, we use the full parallel training data (rather than a small development set) to estimate the contribution of the sparse features in phase 1. Second, sparse features can lead to overfitting. To prevent this from hurting our model's ability to generalize to new data, we do two things. First, we use "grammar and language model folds" (translation grammars and language models built from other portions of the training data than are being used for discriminative training), and second, we run the phase 2 line search on a held-out development set. Finally, since our algorithm requires decoding the entire training corpus, it is desirable (on computational grounds) to only require one or two passes through the training data. To get the most out of these passes, we rescale features by their inverse frequency which improves the scaling of the optimization problem. In addition to learning with few passes through the training data, the HOLS algorithm has the advantage that it is easily parallelizable.

After reviewing related work in the next section, we analyze two obstacles to effective discriminative learning for machine translation: overfitting (since both rules and their weights must be learned, if they are learned together degenerate solutions that fail to generalize are possible) and poor scaling (since MT decoding is so expensive, it is not feasible to make many passes through large amounts of training data, so optimization must be efficient). We then present the details of our algorithm that addresses these issues, give results on three language pairs, and conclude.

2 Related Work

Discriminative training of machine translation systems has been a widely studied problem for the last ten years. The pattern of using small, high-quality development sets to tune a relatively small number of weights was established early (Och and Ney, 2002; Och, 2003). More recently, standard structured prediction algorithms that target linearly decomposable approximations of translation quality metrics have been thoroughly explored (Liang et al., 2006; Smith and Eisner, 2006; Watanabe et al., 2007; Rosti et al., 2010; Hopkins and May, 2011; Chiang, 2012; Gimpel and Smith, 2012; Cherry and Foster, 2012; Saluja et al., 2012). These have without exception used sentence-level approximations of BLEU to determine oracles and update weights using a variety of criteria and with a variety of different theoretical justifications.

Despite advancements in discriminative training for machine translation, large-scale discriminative training with rule indicator features has remained notoriously difficult. **Rule indicator features** are an extremely sparse and expressive parameterization of the translation model: every rule has a feature, each of which has its own separately tuned weight, which count how often a specific rule is used in a translation. Early experiments (Liang et al., 2006) used the structured perceptron to tune a phrase-based system on a large subset of the training data, showing improvements when using rule indicator features, word alignment features, and POS tag features. Another early attempt (Tillmann and Zhang, 2006) used phrase pair and word features in a block SMT system trained using stochastic gradient descent for a convex loss function, but did not compare to MERT. Problems of overfitting and degenerate derivations were tackled with a probabilistic latent variable model (Blunsom et al., 2008) which used rule indicator features yet failed to improve upon the MERT baseline for the standard Hiero features. Techniques for distributed learning and feature selection for the perceptron loss using rule indicator, rule shape, and source side-bigram features have recently been proposed (Simianer et al., 2012), but no comparison to MERT was made.

3 Difficulties in Large-Scale Training

Discriminative training for machine translation is complicated by several factors. First, both translation rules and feature weights are learned from parallel data. If the same data is used for both tasks, overfitting of the weights is very possible.[1] Second, the standard MT cost function, BLEU (Papineni et al., 2002), does not decompose additively over training instances (because of the "brevity penalty") and so approximations are used—these often have problems with the length (Nakov et al., 2012). Finally, state-of-the-art MT systems make extensive good use of "dense" features, such as the log probability of translation decisions under a simpler generative translation model. Our goal is to begin to use much sparser features without abandoning the proven dense features; however, extremely sparse features leads to problems of scaling in the optimization problem as we will show.

3.1 Training Data and Overfitting

One of the big questions in discriminative training of machine translation systems is why standard machine learning techniques can perform so poorly when applied to large-scale learning on the training data. Figure 1 shows a good example of this. The structured SVM (Tsochantaridis et al., 2004; Cherry and Foster, 2012) was used to learn the weights for a Chinese-English Hiero system (Chiang, 2005) with just eight features, using stochastic gradient descent (SGD) for online learning (Bottou, 1998; Bottou, 2010). The weights were initialized from MERT values tuned on a 2k-sentence dev set (MT06), and the figure shows the progress of the online method during a single pass through the 300k-sentence Chinese-English FBIS training set.

As the training progresses in Figure 1, BLEU scores on the training data go up, but scores on the

[1] Previous work has attempted to mitigate the risk of overfitting through careful regularization (Blunsom et al., 2008; Simianer et al., 2012).

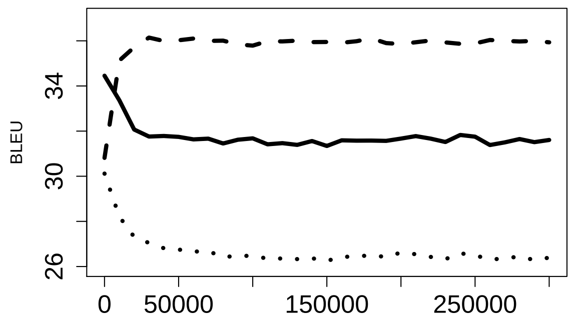


Figure 1: Progress of the online SVM training method after each training instance on FBIS dataset. The solid line is BLEU on the test set, training set is the dashed line, and the dev set is dotted.

dev and test sets go down. If we hope to apply discriminative training techniques for not eight but millions of features on the training data, we must find a way to prevent this overfitting.

We suggest that an important reason why overfitting occurs is that the training data is used not only to tune the system but also to extract the grammar, and the target side is included in the data used to build the language model. To test this hypothesis, we compare tuning using three different dev sets: 1000 sentences from the standard 4-reference MT06 dev set (Dev1000), a random selection of 1000 sentences that overlap with the corpus used to extract translation rules (In1000), and 1000 sentences that came from the training data but were then excluded from rule extraction (Out1000). We run MERT on each of these and evaluate. For evaluation we compare three different sets: a random 1000 sentences from the training corpus that was used to create the grammars but which do not overlap with In1000 (Train1000), the 1000 sentence dev set (Dev1000), and the standard 4-reference MT02-03 test set (Test). The entire experiment (including selection of the 1000 sentences) was replicated 5 times.

Table 1 shows the results, averaging over replications. Out1000 gives much higher scores on the testing data, validating our hypothesis that tuning on data used to build the LM and grammar can lead to overfitting. However, the results also show that tuning on the training data, even when it is held-out, can still lead to a small reduction in translation quality. One possible reason is that, unlike the training data which may come from various domains, the dev data is in the same domain as the test data and is typically of higher quality (e.g., it has multiple references).

Table 1: MERT on Zh-En FBIS

Tuning Set	Train1000	Dev1000	Test
Dev1000	32.2±1.1	30.2±.1	34.1±.3
In1000	37.0±1.2	25.7±.7	30.1±.6
Out1000	34.9±.8	29.0±.4	33.6±.5

3.2 Poor Scaling

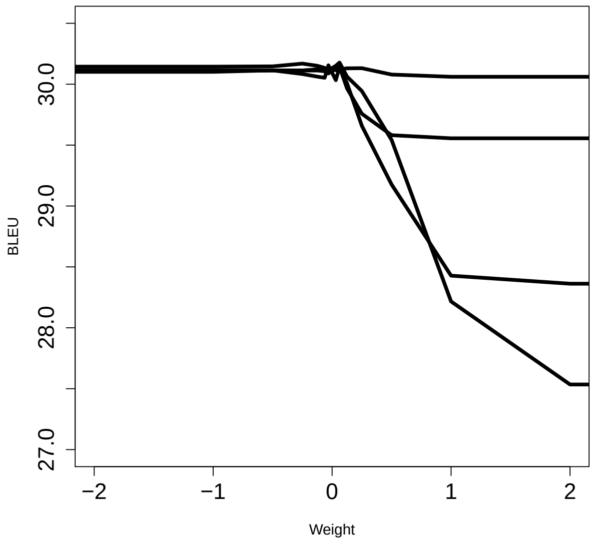
When features occur with different frequencies, changing the weights of more frequent features has a larger effect than changing the weights of less frequent features.[2] An example of frequent features that have a large impact on the translation quality are the language model and translation model features. These features are non-zero for every sentence, and changing their weights slightly has a large impact on translation output. In contrast, changing the weight drastically for a feature that is non-zero for only one out of a million sentences has very little effect on translation metrics. The sensitivity of the translation output to some feature weights over others was also pointed out in a recent paper (Chiang, 2012).

When the objective function is more sensitive in some dimensions than others, the optimization problem is said to be **poorly scaled** (Nocedal and Wright, 2000), and can slow down the convergence rate for some optimizers. A typical fix is to rescale the dimensions, as we will do in Section 5.2.

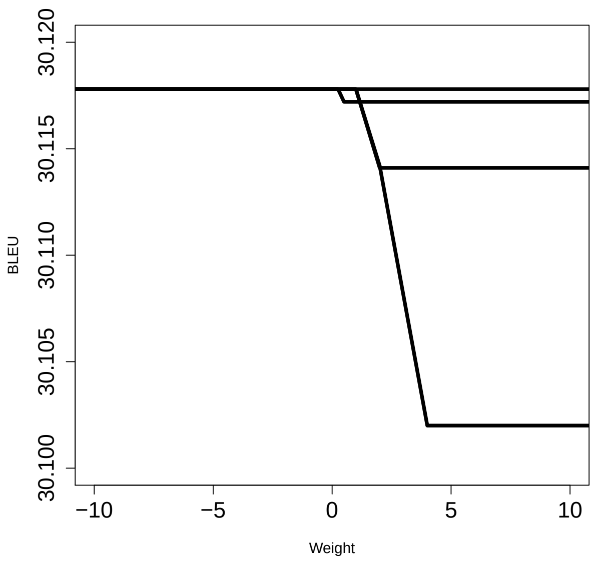
To verify that BLEU is poorly scaled with respect to weights of rule indicator features, we look at the effect of changing the weights for individual rules. We vary the feature weights for four randomly chosen frequent rules and four randomly chosen infrequent rules on our FBIS dev set (Figure 2). One can think of this plot as a "cross-section" of the BLEU score in the direction of the feature weight. The dense features are set to MERT-tuned values which are normalized to one. All other rule indicator features are set to zero, except the rule feature weight that is varied. The frequent features

[2]By the "frequency of a feature" we mean this: given a set of input instances, how many input instances the feature is nonzero in the space of possible outputs for that input.

were selected randomly from the 20 most common rule indictor features in the n-best lists on the dev set, and the infrequent features were selected from the features that only occurred once in these n-best lists. The plots indicate that the BLEU score is



(a) Four representative *frequent* sparse features.



(b) Four representative *infrequent* sparse features

Figure 2: The effect of varying weights for rule indicator features on the BLEU score. Note the difference of scale on the y axis.

poorly scaled for rule feature weights. Changing the weights for one of the common features changes the BLEU score by almost 2.5 BLEU points, while for the infrequent features the BLEU score changes by at most $.02$ BLEU points. We take this as a sign that gradient descent based optimizers for machine translation with rule features could be slow to converge due to poor scaling, and that rescaling will improve convergence.

3.3 Sentence Level Approximations to BLEU

Finally, we note that discriminative training methods often use a sentence level approximation to BLEU. It has been shown that optimizing corpus level BLEU versus sentence level BLEU can lead to improvements of up to nearly $.4$ BLEU points on the test set (Nakov et al., 2012). Possible fixes to this problem include using a proper sentence level metric such a METEOR (Denkowski and Lavie, 2011) or a pseudo-corpus from the last few updates (Chiang et al., 2008). However, in light of the result from section 3.1 that tuning on the dev set is still better than tuning on a held-out portion of the training data, we observe that tuning a corpus level metric on a high-quality dev set from the same domain as the test set probably leads to the best translation quality. Attempts to improve upon this strong baseline lead us to the development of the HOLS algorithm which we describe next.

4 Held-Out Line Search Algorithm

In this section we give the details of the learning algorithm that we developed for use in large-scale discriminative training for machine translation, which we call the Held-Out Line Search algorithm (abbreviated HOLS). It optimizes millions of features using evidence from the full set of parallel training data to obtain optimal predictive performance on a secondary development set.

The learning algorithm is a batch optimizer where each iteration has two phases: a gradient calculation phase and a line search phase. In the gradient calculation phase, a surrogate loss function is used to compute a gradient for the feature weights. The gradient is computed over a subset of the training data. In the line search phase, a separate optimizer (MERT) is used to search along this gradient to optimize the evaluation score of the one-best prediction of a translation system on a secondary development set.[3] The secondary dev set is a crucial aspect of the algorithm that helps reduce overfitting (we will demonstrate this in the experiments section).

During the line search phase we allow some of the feature weights to be adjusted independently of the line search. We will call the features we optimize independently the dense features, and the features we include in the line search the sparse features.[4] The feature vector space V is the direct sum $V = V_d \oplus V_s$, where V_d is the vector space of the dense features and V_s is the vector space of the sparse features. The feature and weight vectors decompose as $\vec{f} = \vec{f}_d + \vec{f}_s$ and $\vec{w} = \vec{w}_d + \vec{w}_s$. $\vec{f}_d$ and $\vec{w}_d$ are in the dense vector space, and the $\vec{f}_s$ and $\vec{w}_s$ are in the sparse vector space.

In the gradient phase, we calculate a gradient of the surrogate loss function and project it onto the subspace of the sparse features. Let P_s be the projection operator onto V_s. Then the gradient projected onto the sparse feature space is

$$\vec{g} = \mathrm{P}_s \nabla_{\vec{w}} \tilde{L}(\vec{w}, \mathcal{D}_g)$$

where $\mathcal{D}_g$ is the subset of the training data used to calculate this gradient, and $\tilde{L}$ is the surrogate loss function. This just sets the dense components of the gradient of $\tilde{L}$ to zero.

In the line search phase, we use a separate optimizer to optimize the weights for the dense features and the stepsize α. Let L be the loss function we wish to minimize, then

$$(\vec{w}_d^*, \alpha^*) = \arg\min_{\vec{w}_d, \alpha} L(\vec{w}_d + \vec{w}_s + \alpha\vec{g}, \mathcal{D}_l)$$

Note $\vec{w}_s$ is held fixed from the previous iteration. $\mathcal{D}_l$ is the portion of the training data which is used in the line search phase, and must not overlap with $\mathcal{D}_g$ used in the gradient calculation phase.[5]

After the line search, the dense weights are updated to $\vec{w}_d^*$, and the sparse weights are updated with $\vec{w}_s \leftarrow \vec{w}_s + \alpha^* \vec{g}$. The process repeats for another iteration as desired (or until convergence).

[3]While we use BLEU any loss function whose sufficient statistics decompose over training instances could be used.

[4]The split over the features does not have to be done this way in practice.

[5]$L(\vec{w}_d^*, \alpha^*, \mathcal{D}_l)$ can be thought of as unbiased or more accurately less biased estimator of expected loss when $\mathcal{D}_l \cap \mathcal{D}_g = \emptyset$.

5 Procedure for Large-Scale Training

Now that we have described the HOLS algorithm in general, we next describe how to apply it to large-scale training of machine translation systems with millions of features. We find that it is necessary to use disjoint sets of training instances for grammar extraction and gradient estimation (§5.1) and to deal with the poor scaling of the optimization problem (§5.2).

5.1 Grammar and Language Model Folds

To address the problem of overfitting on the training data, we split the training data into n-folds, and extract grammars for each fold using the data from the other $n-1$ folds. Similarly, we build a language model for each fold using a target language monolingual corpus and the target side of the training data from the other $n-1$ folds. Whenever we decode a sentence from the training data, we use the grammar and language model for the appropriate fold. This ensures that a sentence is never decoded using a grammar or language model it helped build, thereby reducing the overfitting effect demonstrated in §3.1.

To perform the training, the HOLS algorithm is used on the training data. In our experiments, only 1-2 passes over the training data are necessary for significant gains. Data from one of the grammar folds is used for the line search, and the rest of the training data is used to calculate the gradient.

The procedure is iterative, first decoding training data to obtain a gradient, and then performing a line search with data from a held-out grammar fold. Instead of decoding the whole set of sentences used for the gradient updates at once, one can also decode a portion of the data, do a gradient update, and then continue the next iteration of HOLS on the remaining data before repeating.

The last line search of the HOLS algorithm is done using dev data, rather than training data. This is because the dev data is higher quality, and from Table 1 we can see that tuning on dev data produces better results than tuning on training data (even if the training data has been held out from the grammar process). The initial weights are obtained by running MERT on a subset of the one of the grammar folds.

If one has an existing implementation of an optimizer for the loss function used during the line search (in our case MERT), it can be used to perform the line search. This is done simply by calling MERT with two extra features in addition to the dense features and omitting the sparse features.

To see how, notice that the feature weights during the line search are decomposed as $\vec{w} = \vec{w}_{dense} + \vec{w}_{sparse} + \alpha\vec{g}$ where $\vec{g}$ is in the sparse feature subspace, so the model score decomposes as $score(x,y) = \vec{w}_d \cdot \vec{f_d}(x,y) + \vec{w}_s \cdot \vec{f_s}(x,y) + \alpha\vec{g} \cdot \vec{f_s}(x,y)$ where x is the input translation, y is the output translation and derivation. If we create two new features $f_1(x,y) = \vec{w}_s \cdot \vec{f_s}(x,y)$ and $f_2(x,y) = \vec{g} \cdot \vec{f_s}(x,y)$ then the score can be written

$$\begin{aligned} score(x,y) &= \vec{w}_d \cdot \vec{f_d}(x,y) \\ &\quad + f_1(x,y) + \alpha f_2(x,y) \\ &= (\vec{w}_d, 1, \alpha) \cdot (\vec{f_d}, f_1, f_2) \end{aligned}$$

Thus we can do the line search simply by calling MERT with the features $(\vec{f_d}, f_1, f_2)$. [6]

In summary our training algorithm is as follows: 1) split the training data into n-folds (we use $n = 5$), 2) initialize the dense weights to MERT values, 3) decode some or all the data in 4 of the 5 folds to get a gradient, 4) condition as in §5.2 (see below), 5) run MERT on a 10k subset of the remaining fold to do the line search, 6) repeat steps 3-4 until convergence or stop as desired, and 7) run MERT on the normal dev set as a final step. We only run MERT on a 10k subset of one of the folds so it does not require running MERT on an entire fold.

In the special case where just one iteration of HOLS is performed, the procedure is very simple: decode the training data to get a gradient, include the components of the gradient as an extra feature f_2 in addition to the dense features, and tune on a dev set using MERT.

5.2 Conditioning

To address the problem of poor scaling, we use a simple strategy of rescaling each component of the gradient based on how frequent the feature is. We call this process "conditioning." For each feature, we simply divide the corresponding dimension of the gradient by the number of n-best lists in which the feature was non-zero in.

The necessity for conditioning is evident when we run the HOLS algorithm as detailed so far on the training data without conditioning. On subsequent iterations, we observe that the features with the highest component of the gradient oscillate between iterations, but the rest of the feature gradients stay the same.

Based on our knowledge that the optimization problem was poorly scaled, we divided by the frequency of the feature. We can give the following heuristic justification for our method of conditioning. For the ith feature weight, we will take a step Δw_i. Assume that we want to take the step Δw_i proportional to the average gradient $\hat{g}_i$ calculated from n-best lists in which the feature is non-zero. In other words, we want $\Delta w_i = \alpha \hat{g}_i$. Let g_i be the total gradient calculated by adding the gradients over all n-best lists (i.e. summing over training examples in the corpus). For a feature that is nonzero in exactly n_i n-best lists, the gradient from each example will have been added up n_i times, so the total gradient $g_i = n_i \hat{g}_i$. Therefore we should take the step $\Delta w_i = \alpha g_i / n_i$. In other words, we rescale each component g_i of the gradient by $1/n_i$ before taking the gradient step.

We can relate this argument back to the oscillation we observed of the rule feature weights. For rules that are used a thousand times more often than the average rule, the corresponding component of the gradient is roughly a thousand times larger. But that does not indicate that the adjustment Δw_i to the rule weight should be a thousand times larger in each iteration.

6 Experiments

We evaluate and analyze the performance of our training method with three sets of experiments. The first set of experiments compares HOLS to other tuning algorithms used in machine translation in a medium-scale discriminative setting. The second set looks in detail at HOLS for large scale discriminative training for a Chinese-English task. The third set looks at two other languages.

All the experiments use a Hiero MT system with rule indicator features for the sparse features and the

[6]We could constrain the weight for f_1 to be 1, but this is not necessary since since MERT is invariant to the overall scale of the weights.

Table 2: Corpora

Language	Corpus	Sentences	Tokens	
			Source	Target
En	Gigaword	24M		594M
Ar-En	Train	1M	7M	31M
	Dev (MT06)	1797	13K	236K
	MT05	1,056	7K	144K
	MT08nw	813	5K	116K
	MT05wb	547	5K	89K
Mg-En	Train	89K	2.1M	1.7M
	Dev	1,359	34K	28K
	Test	1,133	29K	24K
Zh-En	Train (FBIS)	302K	1M	9.3M
	Dev (MT06)	1,664	4K	192K
	Test (MT02-03)	1,797	5K	223K
	MT08	1,357	4K	167K

following 8 dense features: LM, phrasal and lexical $p(e|f)$ and $p(f|e)$, phrase and word penalties, and glue rule. The total number of features is 2.2M (Mg-En), 28.8M (Ar-En), and 10.8M (Zh-En). The same features are used for all tuning methods, except MERT baseline which uses only dense features. Although we extract different grammars from various subsets of the training corpus, word alignments were done using the entire training corpus. We use GIZA++ for word alignments (Och and Ney, 2003), Thrax (Weese et al., 2011) to extract the grammars, our decoder is cdec (Dyer et al., 2010) which uses KenLM (Heafield, 2011), and we used a 4-gram LM built using SRILM (Stolcke, 2002). Our optimizer uses code implemented in the pycdec python interface to cdec (Chahuneau et al., 2012). To speed up decoding, for each source RHS we filtered the grammars to the top 15 rules ranked by $p(e \mid f)$. Statistics about the datasets we used are listed in Table 2.

We use the "soft ramp 3" loss function (Gimpel, 2012; Gimpel and Smith, 2012) as the surrogate loss function for calculating the gradient in HOLS. It is defined as

$$\tilde{L} = \sum_{i=1}^{n} \Big[-\log \sum_{y \in \text{Gen}(x_i)} e^{\vec{w}\cdot\vec{f}(x_i,y) - cost(y_i,y)} + \log \sum_{y \in \text{Gen}(x_i)} e^{\vec{w}\cdot\vec{f}(x_i,y) + cost(y_i,y)} \Big]$$

where the sum over i ranges over training examples, $\text{Gen}(x)$ is the space of possible outputs and derivations for the input x, and $cost(y_i, y)$ is add one smoothing sentence level BLEU.[7]

Except where noted, all experiments are repeated 5 times and results are averaged, initial weights for the dense features are drawn from a standard normal, and initial weights for the sparse features are set to zero. We evaluate using MultEval (Clark et al., 2011) and report standard deviations across optimizer runs and significance at $p = .05$ using MultEval's built-in permutation test. In the large-scale experiments for HOLS, we only run the full optimizer once, and report standard deviations using multiple runs of the last MERT run (i.e. the last line search on the dev data).

6.1 Comparison Experiments for ZH-EN

Our first set of experiments compares the performance of the proposed HOLS algorithm to learning algorithms popularly used in machine translation on a Chinese-English task. We also compare to a close relative of the HOLS algorithm: optimizing the soft ramp 3 loss directly with online stochastic gradient descent and with conditioning. As we will see, SGD SOFTRAMP3 performs significantly worse than HOLS, despite both algorithms optimizing similar loss functions.

In the experiments in this section, we do not use the full version of the training setup described in §5 since we wish to compare to algorithms that do not necessarily scale to large amounts of training data. We therefore use only one fifth of the training data for learning the weights for both the dense and sparse features.

In this section we refer to the subset of the training data used to learn the weights as the **tuning set** (Tune). The grammar and LM are built using the training data that is not in the tuning set (the LM also includes the English monolingual corpus), and the weights for the features are tuned using the tuning set. This is similar to the typical train-dev-test split commonly used to tune machine translation systems, except that the tuning set is much larger (60k sentence pairs versus the usual 1k-2k) and comes from a random subset of the training data rather than a

[7]We found this loss function to work well, but other "soft" loss functions (Gimpel, 2012; Gimpel and Smith, 2012) also work. Gen(x) is restricted to a k-best size of 1000. Following (Gimpel, 2012) $cost(y_i, y)$ is multiplied by a factor of 20.

Table 3: Comparison Experiments for Zh-En

Algorithm	Tune	MT08	Runtime
MERT	$22.1_{\pm .1}$	$23.1_{\pm .1}$	**6 hours**
PRO	$23.8_{\pm .05}$	$\mathbf{23.6}_{\pm .1}$	2 weeks
MIRA	$21.7_{\pm .1}$	$22.5_{\pm .1}$	19 hours
SOFTRAMP3	$21.5_{\pm .3}$	$22.3_{\pm .3}$	29 hours
HOLS	$22.3_{\pm .1}$	$23.4_{\pm .1}$	10 hours
HILS	$\mathbf{24.3}_{\pm .2}$	$22.4_{\pm .1}$	10 hours

specialized development set.

We compare MERT, PRO (Hopkins and May, 2011), MIRA (Chiang, 2012), SOFTRAMP3, HOLS, and a variant of HOLS which we call HILS (discussed below). For HOLS, we used 10k of the 60k tuning set for the line search, and the rest of the tuning set was used for calculating the gradient. For HILS ("Held-In" Line Search), the full 60k tuning set was used to calculate the gradient, but the line search was on a 10k subset of that set. For MERT, we used a 10k subset of the tuning data because it takes a long time to run on large datasets, and it only has the eight dense features and so does not need the entire 60k tuning set. All the subsets are drawn randomly. Conditioning was performed only for HOLS, HILS, and SOFTRAMP3 because conditioning would affect the regularizer for PRO and require modifications to the MIRA algorithm. To do the conditioning for SOFTRAMP3 we used rule count during extraction of the grammar and not the frequency in the n-best lists because the online nature of SOFTRAMP3 prevents us from knowing how frequent a rule will be (and the dense features are conditioned using the corpus size). We chose MIRA's best learning rate ($\eta = .001$) from $\{.1, .01, .001\}$, used default settings for PRO in cdec, and for SOFTRAMP3 we used the same loss function as HOLS but included an L_2 regularizer of strength .001 and used a stepsize of 1 (which was scaled because of conditioning). To remedy problems of length bias for sentence level BLEU, we used brevity penalty smoothed and grounded BLEU+1 for sentence level scores (Nakov et al., 2012). Tuning was repeated four times with different initial weights, except for PRO which we only ran three times (due to training costs). The initial weights for MERT were drawn from a standard normal distribution, and final MERT weights were used as the initial weights for the dense features for the other algorithms. Initial weights for the sparse features were set to zero. For HOLS, and HILS, tuning set BLEU scores were evaluated on the set that the line search was run on. We also report run times for 8 threads on an Opteron 6220 processor.[8]

The results are shown in Table 3. PRO and HOLS are a statistically significant improvement upon the MERT baseline on the MT08 test data, but MIRA, SOFTRAMP3, and HILS are not.

HILS dramatically overfits the tuning set, while HOLS does not, justifying the use of a held-out dataset for the line search. SOFTRAMP3 performs significantly worse than HOLS on the test set. PRO is a promising training algorithm, but does not scale to the full FBIS corpus because it requires many iterations.

6.2 Full ZH-EN and Ablation Experiments

This set of experiments evaluates the performance of the full HOLS algorithm described in §5 for large-scale discriminative training on the full FBIS Chinese-English dataset. Since this is a relatively small and widely studied dataset, we also investigate what happens if different aspects of the procedure are omitted.

Table 4 gives the results. The number of updates is the number of times the HOLS line search optimizer is run (gradient updates). For 2 passes, 4 updates, a line search is performed after a half pass through the training data, which is repeated four times for a total of two passes.

Using just one pass through the training data and

[8]Standard MIRA and SGD SOFTRAMP3 are not parallelizable and only use a single thread. All of these algorithms were run for one iteration, except for MERT which ran for at least seven iterations, and PRO which we stopped after 20 iterations.

Table 4: Full-scale Chinese-English and Ablation Experiments

Configuration	Dev	Test
MERT Baseline	$29.9_{\pm .3}$	$34.0_{\pm .8}$
2 Pass, 4 updates	$\mathbf{31.1}_{\pm .2}$	$\mathbf{35.1}_{\pm .4}$
1 Pass, 1 update	$30.7_{\pm .1}$	$34.6_{\pm .5}$
−Folds	$30.0_{\pm .2}$	$34.0_{\pm .4}$
−Conditioning	$30.1_{\pm .1}$	$34.2_{\pm .2}$

Table 5: Arabic-English

System	Dev (MT06)	MT05	MT08(nw)	MT08(wb)
MERT Baseline	39.2±.4	50.3±.4	45.2±.2	29.4±.14
HOLS 1 Pass, 2 updates	**39.9**±.9	**51.2**±.4	**45.8**±.4	**30.0**±.4
ΔBLEU	+.7	+.9	+.6	+.6

Table 6: Malagasy-English

System	Dev	Test
MERT Baseline	19.8±.3	17.7±.2
HOLS 1 Pass, 1 update	**20.5**±.1	**18.4**±.2
ΔBLEU	+.7	+.7

one gradient update, HOLS improves upon the MERT baseline by .6 BLEU points, which is a statistically significant improvement. With 2 passes through the training data and 4 gradient updates, HOLS performs even better, obtaining a 1.1 BLEU point improvement over the baseline and is also statistically significant. With 16 threads, 1 pass, 1 update completed in 9 hours, and 2 pass, 4 updates, completed in 40 hours. The medium-scale PRO setup in §6.1 obtains a result of $34.4 \pm .1$ on this test set, which is a statistically significant improvement of .4 BLEU points over the MERT baseline but does not beat the large-scale HOLS results.

Is folding and conditioning necessary? We experiment with what happens if grammar and LM folds are not used and if conditioning is not done. −Folds denotes 1 pass 1 update without folds, and −Conditioning denotes 1 pass 1 update without conditioning. We can see that both these steps are important for the training procedure to work well.

The decrease in performance of the training procedure without folds or conditioning is dramatic but not too surprising. With just one gradient update, one would expect conditioning to be very important. And from the lessons learned in section 3.1, one would also expect the procedure to perform poorly or even worse than the MERT baseline without grammar or LM folds. But because HOLS runs MERT on the dev data for the last line search, it is almost impossible for HOLS to be worse than the MERT baseline. (This, in fact, was part of our motivation when we originally attempted the HOLS algorithm.)

6.3 Other Language Pairs

The last set of experiments looks at the performance of the learning algorithm for two other languages and data scenarios for one pass through the training data. Using the same setup for large-scale discriminative training as before, we apply the training procedure to a large data scenario Arabic-English task and a small data scenario Malagasy-English task (Tables 5 and 6). The training procedure gives statistically significant improvements over the baseline by .6 to .9 BLEU for Arabic, and a statistically significant improvement of .7 BLEU for Malagasy. With 16 threads, the runtime was 44 hours for Arabic and 5 hours for Malagasy.

7 Conclusion

We have explored the difficulties encountered in large-scale discriminative training for machine translation, and introduced a learning procedure designed to overcome them and scale to large corpora. We leave to future work to experiment with feature sets designed for the large-scale discriminative setting. In particular, we hope this framework will facilitate incorporation of richer linguistic knowledge into machine translation.

Acknowledgments

This work was sponsored by the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number W911NF-10-1-0533. Jeffrey Flanigan would like to thank his co-advisor Lori Levin for support and encouragement during this work.

References

Phil Blunsom, Trevor Cohn, and Miles Osborne. 2008. A discriminative latent variable model for statistical machine translation. In *Proc. ACL-HLT*.

Léon Bottou. 1998. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning* *and Neural Networks*. Cambridge University Press, Cambridge, UK. revised, oct 2012.

Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August. Springer.

V. Chahuneau, N. A. Smith, and C. Dyer. 2012. pycdec: A python interface to cdec. *The Prague Bulletin of Mathematical Linguistics*, 98:51–61.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proc. of NAACL*.

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

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *In ACL*, pages 263–270.

David Chiang. 2012. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, pages 1159–1187.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pages 176–181, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michael Denkowski and Alon Lavie. 2011. Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the EMNLP 2011 Workshop on Statistical Machine Translation*.

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of ACL*.

Kevin Gimpel and Noah A. Smith. 2012. Structured ramp loss minimization for machine translation. In *Proc. of NAACL*.

K. Gimpel. 2012. *Discriminative Feature-Rich Modeling for Syntax-Based Machine Translation*. Ph.D. thesis, Carnegie Mellon University.

Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, Edinburgh, UK, July. Association for Computational Linguistics.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proc. of EMNLP*.

Percy Liang, Alexandre Bouchard-côté, Dan Klein, and Ben Taskar. 2006. An end-to-end discriminative approach to machine translation. In *In Proceedings of the Joint International Conference on Computational Linguistics and Association of Computational Linguistics (COLING/ACL*, pages 761–768.

Preslav Nakov, Francisco Guzmán, and Stephan Vogel. 2012. Optimizing for sentence-level bleu+1 yields short translations. In Martin Kay and Christian Boitet, editors, *COLING*, pages 1979–1994. Indian Institute of Technology Bombay.

Jorge Nocedal and Stephen J. Wright. 2000. *Numerical Optimization*. Springer.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proc. of ACL*.

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

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. of ACL*.

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 Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Antti-Veiko Rosti, Bing Zhang, Spyros Matsoukas, and Richard Schwartz. 2010. BBN system description for WMT10 system combination task. In *Proc. WMT*.

Avneesh Saluja, Ian Lane, and Joy Zhang. 2012. Machine Translation with Binary Feedback: a large-margin approach. In *Proceedings of The Tenth Biennial Conference of the Association for Machine Translation in the Americas*, San Diego, CA, July.

Patrick Simianer, Chris Dyer, and Stefan Riezler. 2012. Joint feature selection in distributed stochastic learning for large-scale discriminative training in SMT. In *Proc. ACL*.

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proc. of ACL*.

Andreas Stolcke. 2002. Srilm - an extensible language modeling toolkit. pages 901–904.

Christoph Tillmann and Tong Zhang. 2006. A discriminative global training algorithm for statistical mt. In *Proceedings of the 21st International Conference on*

Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, ACL-44, pages 721–728, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 104–, New York, NY, USA. ACM.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proc. EMNLP-CoNLL*.

Jonathan Weese, Juri Ganitkevitch, Chris Callison-Burch, Matt Post, and Adam Lopez. 2011. Joshua 3.0: syntax-based machine translation with the thrax grammar extractor. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, WMT '11, pages 478–484, Stroudsburg, PA, USA. Association for Computational Linguistics.