Journal of Machine Learning Research 13 (2012) 1159–1187 Submitted 7/11; Revised 2/12; Published 4/12

Hope and fear for discriminative training of statistical translation models

David Chiang CHIANG@ISI.EDU
USC Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292, USA

Editor: Michael Collins

Abstract

In machine translation, discriminative models have almost entirely supplanted the classical noisy-channel model, but are standardly trained using a method that is reliable only in low-dimensional spaces. Two strands of research have tried to adapt more scalable discriminative training methods to machine translation: the first uses log-linear probability models and either maximum likelihood or minimum risk, and the other uses linear models and large-margin methods. Here, we provide an overview of the latter. We compare several learning algorithms and describe in detail some novel extensions suited to properties of the translation task: no single correct output, a large space of structured outputs, and slow inference. We present experimental results on a large-scale Arabic-English translation task, demonstrating large gains in translation accuracy.

Keywords: machine translation, structured prediction, large-margin methods, online learning, distributed computing

1. Introduction

Statistical machine translation (MT) aims to learn models that can predict, given some utterance in a source language, the best translation into some target language. The earliest of these models were generative (Brown et al., 1993; Och et al., 1999): drawing on the insight of Warren Weaver in 1947 that "translation could conceivably be treated as a problem in cryptography" (Locke and Booth, 1955), they treated translation as the inverse of a process in which target-language utterances are generated by a *language model* and then changed into source-language utterances via a noisy channel, the *translation model*.

Och and Ney (2002) first proposed evolving this noisy-channel model into a discriminative log-linear model, which incorporated the language model and translation model as features. This allowed the language model and translation model be to scaled by different factors, and allowed the addition of features beyond these two. Although discriminative models were initially trained by maximum-likelihood estimation, the method that quickly became dominant was minimum-error-rate training or MERT, which directly minimizes some loss function (Och, 2003). The loss function of choice is most often Bleu (rather, 1 − Bleu), which is the standard metric of translation quality used in current MT research (Papineni et al., 2002). However, because this loss function is in general non-convex and non-smooth, MERT tends to be reliable for only a few dozen features.

Two strands of research have tried to adapt more scalable discriminative training methods to machine translation. The first uses log-linear probability models, as in the original work of Och and Ney (2002), either continuing with maximum likelihood (Tillmann and Zhang, 2006; Blunsom et al., 2008) or replacing it with minimum risk, that is, expected loss (Smith and Eisner, 2006; Zens et al., 2008; Li and Eisner, 2009; Arun et al., 2010). The other uses linear models and large-margin methods (Liang et al., 2006; Watanabe et al., 2007; Arun and Koehn, 2007); we have followed this approach (Chiang et al., 2008b) and used it successfully with many different kinds of features (Chiang et al., 2009; Chiang, 2010; Chiang et al., 2011).

Here, we provide an overview of large-margin methods applied to machine translation, and describe in detail our approach. We compare MERT and minimum-risk against several online large-margin methods: stochastic gradient descent, the Margin Infused Relaxed Algorithm or MIRA (Crammer and Singer, 2003), and Adaptive Regularization of Weights or AROW (Crammer et al., 2009). Using some simple lexical features, the best of these methods, AROW, yields a sizable improvement of 2.4 BLEU over MERT in a large-scale Arabic-English translation task.

We discuss three novel extensions of these algorithms that adapt them to particular properties of the translation task. *First*, in translation, there is no single correct output, but only a *reference* translation, which is one of many correct outputs. We find that training the model to generate the reference exactly can be too brittle; instead, we propose to update the model towards *hope* translations which compromise between the reference translation and translations that are easier for the model to generate (Section 4). *Second*, translation involves a large space of structured outputs. We try to efficiently make use of this whole space, like most recent work in structured prediction, but unlike much work in statistical MT, which relies on n-best lists of translations instead (Section 5). *Third*, inference in translation tends to be very slow. Therefore, we investigate methods for parallelizing training, and demonstrate a novel method that is expensive, but highly effective (Section 6).

2. Preliminaries

In this section, we outline some basic concepts and notation needed for the remainder of the paper. Most of this material is well-known in the MT literature; only Section 2.4, which defines the loss function, contains new material.

2.1 Setting

In this paper, models are defined over *derivations* d, which are objects that encapsulate an input sentence $f(d)$, an output sentence $e(d)$, and possibly other information.[1] For any input sentence f, let $\mathcal{D}(f)$ be the set of all valid derivations d such that $f(d) = f$.

A model comprises a mapping from derivations d to feature vectors $\mathbf{h}(d)$, together with a vector of feature weights $\mathbf{w}$, which are to be learned. The model score of a derivation d is $\mathbf{w} \cdot \mathbf{h}(d)$. The 1-best or Viterbi derivation of f_i is $\hat{d} = \arg\max_{d \in \mathcal{D}(f_i)} \mathbf{w} \cdot \mathbf{h}(d)$, and the 1-best or Viterbi translation is $\hat{e} = e(\hat{d})$.

We are given a training corpus of input sentences $f_1, \ldots, f_N$, and reference output translations $e_1, \ldots, e_N$ produced by a human translator. Each e_i is not the only correct translation of f_i, but only one of many. For this reason, often multiple reference translations are available for each f_i, but

1. The variables f and e stand for French and English, respectively, in reference to the original work of Brown et al. (1993).

for notational simplicity, we generally assume a single reference, and describe how to extend to multiple references when necessary.

Note that although the model is defined over derivations, only sentence pairs (f_i, e_i) are observed. There may be more than one derivation of e_i, or there may be no derivations. Nevertheless, assume for the moment that we can choose a *reference derivation* d_i that derives e_i; we discuss various ways of choosing d_i in Section 4.

2.2 Derivation forests

The methods described in this paper should work with a wide variety of translation models, but, for concreteness, we assume a model defined using a weighted synchronous context-free grammar or related formalism (Chiang, 2007). We do not provide a full definition here, but only enough to explain the algorithms in this paper. In models of this type, derivations can be thought of as trees, and the set of derivations $\mathcal{D}(f)$ is called a *forest*. Although its cardinality can be worse than exponential in $|f|$, it can be represented as a polynomial-sized hypergraph $G = (V, E, r)$, where V is a set of nodes, $r \in V$ is the root node, and $E \subseteq V \times V^*$ is a set of hyperedges. We write a hyperedge as $(v \to \mathbf{v})$. A derivation d is represented by an edge-induced subgraph of G such that $r \in d$ and, for every node $v \in d$, there is exactly one hyperedge $(v \to \mathbf{v})$.

We require that $\mathbf{h}$ (and therefore $\mathbf{w} \cdot \mathbf{h}$) decomposes additively onto hyperedges, that is, $\mathbf{h}$ can be extended to hyperedges such that

$$\mathbf{h}(d) = \sum_{(v\to\mathbf{v})\in d} \mathbf{h}(v \to \mathbf{v})$$

This allows us to find the Viterbi derivation efficiently using dynamic programming.

2.3 Bleu

The standard metric for MT evaluation is currently Bleu (Papineni et al., 2002). Since we use this metric not only for evaluation but during learning, it is necessary to describe it in detail.

For any string e, let $g_k(e)$ be the multiset of all k-grams of e. Let K be the maximum size k-grams we will consider; $K = 4$ is standard. For any multiset A, let $\#_A(x)$ be the multiplicity of x in A, let $|A| = \sum_x \#_A(x)$, and define the multisets $A \cap B$, $A \cup B$, and A^* such that

$$\begin{aligned}
\#_{A\cap B}(x) &= \min(\#_A(x), \#_B(x)) \\
\#_{A\cup B}(x) &= \max(\#_A(x), \#_B(x)) \\
\#_{A^*}(x) &= \begin{cases} \infty & \text{if } \#_A(x) > 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

Let c be the candidate translation to be evaluated and let r be the reference translation. Then define a vector of *component scores*

$$\mathbf{b}(c, r) = [m_1, \ldots m_K, n_1, \ldots n_K, \rho]$$

where

$$\begin{aligned}
m_k &= |g_k(c) \cap g_k(r)| \\
n_k &= |g_k(c)| \\
\rho &= |r|
\end{aligned}$$

If there is set of multiple references R, then

$$m_k = \left| g_k(c) \cap \bigcup_{r \in R} g_k(r) \right| \tag{1}$$

$$\rho = \arg\min_{r \in R} \big| |r| - |c| \big| \tag{2}$$

where ties are resolved by letting ρ be the length of the shorter reference.

The component scores are additive, that is, the component score vector for a set of sentences $c_1, \ldots, c_N$ with references $r_1, \ldots, r_N$ is $\sum_i \mathbf{b}(c_i, r_i)$. Then the BLEU score is defined in terms of the component scores:

$$\text{BLEU}(\mathbf{b}) = \exp\left(\frac{1}{K} \sum_{k=1}^{K} \log \frac{m_k}{n_k} + \min\left(0, 1 - \frac{\rho}{n_1}\right) \right)$$

2.4 Loss function

Our learning algorithms assume a loss function $\ell_i(e, e_i)$ that indicates how bad it is to guess e instead of the reference e_i. Our loss function is based on BLEU, but because our learning algorithms are online, we need to be able to evaluate the loss for a single sentence, whereas BLEU was designed to be used on whole datasets. If we try to compute it on a single sentence, several problems arise. If n_k is zero, the BLEU score is undefined; if any of the m_k are zero, the whole BLEU score is zero. Even barring such problems, a BLEU score for a single sentence may not accurately reflect the impact of that sentence on the whole test set (Chiang et al., 2008a).

The standard solution to these problems is to add pseudocounts (Lin and Och, 2004):

$$\text{BLEU}(\overline{\mathbf{b}} + \mathbf{b}) = \exp\left(\frac{1}{K} \sum_{k=1}^{K} \log \frac{\overline{m}_k + m_k}{\overline{n}_k + n_k} + \min\left(0, 1 - \frac{\overline{\rho} + \rho}{\overline{n}_1 + n_1}\right) \right)$$

where $\overline{\mathbf{b}} = [\overline{m}_1, \ldots, \overline{m}_K, \overline{n}_1, \ldots, \overline{n}_K, \overline{\rho}]$ are pseudocounts that must be set appropriately.

Watanabe et al. (2007) score a sentence in the context of all previously seen 1-best translations, which they call the *oracle document*. We follow this approach here, but in order to reduce dependence on the distant past, we use an exponential decay. That is, after processing each training example (f_i, e_i), we update the oracle document using the 1-best translation $\hat{e}$:

$$\overline{\mathbf{b}} \leftarrow 0.9 \cdot (\overline{\mathbf{b}} + \mathbf{b}(\hat{e}, e_i))$$

Then we define a per-sentence metric B that measures the impact that adding a new input and output sentence will have on the BLEU score of the oracle document:

$$B(\mathbf{b}) = \overline{n}_1 \cdot \left(\text{BLEU}(\overline{\mathbf{b}} + \mathbf{b}) - \text{BLEU}(\overline{\mathbf{b}}) \right) \tag{3}$$

The reason for the scaling factor $\overline{n}_1$, which is the size of the oracle document, is to try to correct for the fact that if the oracle document is small, then adding a new sentence will have a large effect on its BLEU score, and vice versa.

Finally, we can define the loss of a translation e relative to e' as the difference between their B scores, following Watanabe et al. (2007):

$$\ell_i(e, e') = B(\mathbf{b}(e', e_i)) - B(\mathbf{b}(e, e_i))$$

and, as shorthand,

$$\ell_i(d, e') \equiv \ell_i(e(d), e')$$
$$\ell_i(d, d') \equiv \ell_i(e(d), e(d'))$$

3. Learning algorithms

In large-margin methods, we want to ensure that the difference, or *margin*, between the correct label and an incorrect label exceeds some minimum; in *margin scaling* (Crammer and Singer, 2003), this minimum is equal to the loss. That is, our learning problem is to minimize:

$$L(\mathbf{w}) = \frac{1}{N} \sum_i L_i(\mathbf{w}) \tag{4}$$

where

$$L_i(\mathbf{w}) = \max_{d \in \mathcal{D}(f_i)} v_i(\mathbf{w}, d, d_i)$$
$$v_i(\mathbf{w}, d, d_i) = \ell_i(d, d_i) - \mathbf{w} \cdot (\mathbf{h}(d_i) - \mathbf{h}(d))$$

Note that since $d_i \in \mathcal{D}(f_i)$ and $v_i(\mathbf{w}, d_i, d_i) = 0$, $L_i(\mathbf{w})$ is always nonnegative. We now review the derivations of several existing algorithms for optimizing (4) for structured models.

3.1 Stochastic gradient descent

An easy way to optimize the objective function $L(\mathbf{w})$ is stochastic (sub)gradient descent (SGD) (Ratliff et al., 2006; Shalev-Shwartz et al., 2007). In SGD, we consider one component $L_i(\mathbf{w})$ of the objective function at a time and update $\mathbf{w}$ by the subgradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_i(\mathbf{w}) \tag{5}$$
$$\nabla L_i(\mathbf{w}) = -(\mathbf{h}(d_i) - \mathbf{h}(d^+))$$

where

$$d^+ = \arg\max_{d \in \mathcal{D}(f_i)} v_i(\mathbf{w}, d, d_i)$$

If, as an approximation, we restrict $\mathcal{D}(f_i)$ to just the 1-best derivation of f_i, then we get the structured perceptron algorithm (Rosenblatt, 1958; Freund and Schapire, 1999; Collins, 2002). Otherwise, we get Algorithm 1. Note that, as is common practice with the perceptron, the final weight vector is the average of the weight vector at each iteration. (Line 6 as implemented here can be inefficient; in practice, we use the trick of Daumé (2006, p. 19) to average efficiently.)

The derivation d^+ is the worst violator of our constraint that the margin be greater than or equal to the loss, and appears frequently in large-margin learning algorithms. We call d^+ the *fear derivation*.[2] An easy way to approximate the fear derivation would be to generate an n-best list and select the derivation from it that maximizes v_i. In Section 5 we discuss better ways to search for the fear derivation.

2. The terminology of *fear* derivations and *hope* derivations to be defined below are due to Kevin Knight.

Algorithm 1 Stochastic gradient descent

```
Require: training examples $(f_1, e_1), \ldots, (f_N, e_N)$
 1: $\mathbf{w} \leftarrow \mathbf{0}$
 2: $\mathbf{s} \leftarrow \mathbf{0}, t \leftarrow 0$
 3: while not converged do
 4:     for $i \in \{1, \ldots, N\}$ in random order do
 5:         UpdateWeights($\mathbf{w}, i$)
 6:         $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{w}$
 7:         $t \leftarrow t + 1$
 8: $\mathbf{w} \leftarrow \mathbf{s}/t$

 9: procedure UpdateWeights($\mathbf{w}$, i)
10:     $d^+ \leftarrow \arg\max_{d \in \mathcal{D}(f_i)} v_i(\mathbf{w}, d, d_i)$
11:     $\mathbf{w} \leftarrow \mathbf{w} + \eta(\mathbf{h}(d_i) - \mathbf{h}(d^+))$
```

3.2 MIRA

Kivinen and Warmuth (1996) derive SGD from the following update:

$$\mathbf{w} \leftarrow \arg\min_{\mathbf{w}'} \left(\frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}\|^2 + L_i(\mathbf{w}') \right) \tag{6}$$

where the first term, the *conservativity* term, prevents us from moving too far in a single iteration. Taking partial derivatives and setting to zero, we get

$$\mathbf{w}' - \mathbf{w} + \eta \nabla L_i(\mathbf{w}') = 0$$

If we make the approximation $\nabla L_i(\mathbf{w}') \approx \nabla L_i(\mathbf{w})$, we get the gradient-descent update again:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L_i(\mathbf{w})$$

But the advantage of using (6) without approximation is that it will not overshoot the optimum if the step size η happens to be too large. This is the Margin Infused Relaxed Algorithm (MIRA) of Crammer and Singer (2003).

The MIRA update (6) replaces the procedure UpdateWeights in Algorithm 1. It is more commonly presented as a quadratic program (QP):

$$\begin{array}{ll} \text{minimize} & \frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}\|^2 + \xi_i \\ \text{subject to} & v_i(\mathbf{w}', d, d_i) - \xi_i \le 0 \qquad \forall d \in \mathcal{D}(f_i) \end{array}$$

where ξ_i is a slack variable.[3] (Note that $\xi_i \ge 0$ since $d_i \in \mathcal{D}(f_i)$ and $v_i(\mathbf{w}', d_i, d_i) = 0$.) The Lagrangian is:

$$\mathcal{L} = \frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}\|^2 + \xi_i + \sum_{d \in \mathcal{D}(f_i)} \alpha_d (v_i(\mathbf{w}', d, d_i) - \xi_i)$$

3. Watanabe et al. (2007) use a different slack variable ξ_{id} for each hypothesis d, which leads to a different update than the one derived below.

Setting partial derivatives to zero gives:

$$\mathbf{w}' = \mathbf{w} + \eta \sum_{d \in \mathcal{D}(f_i)} \alpha_d (\mathbf{h}(d_i) - \mathbf{h}(d))$$

$$\sum_{d \in \mathcal{D}(f_i)} \alpha_d = 1$$

Substituting back into (3.2), we get the following dual problem:

$$\begin{aligned} \text{maximize} \quad & -\frac{\eta}{2} \left\| \sum_{d \in \mathcal{D}(f_i)} \alpha_d (\mathbf{h}(d_i) - \mathbf{h}(d)) \right\|^2 + \sum_{d \in \mathcal{D}(f_i)} \alpha_d v_i(\mathbf{w}, d, d_i) \\ \text{subject to} \quad & \sum_{d \in \mathcal{D}(f_i)} \alpha_d = 1 \\ & \alpha_d \geq 0 \qquad \forall d \in \mathcal{D}(f_i) \end{aligned}$$

In machine translation, and in structured prediction in general, the number of hypotheses in $\mathcal{D}(f_i)$, and therefore the number of constraints in the QP, can be exponential or worse. Watanabe et al. (2007) use the 1 best or 10 best hypotheses. In an earlier version of this work (Chiang et al., 2008b), we used the top 10 fear derivations.[4] Here, we use the cutting-plane algorithm of Tsochantaridis et al. (2004), which repeatedly recomputes the fear derivation and adds it to a working set $\mathcal{S}_i$ of derivations on which the QP is optimized (Algorithm 2). A new fear derivation is added to the working set only if it is a worse violator by a certain margin (ϵ); otherwise, the algorithm terminates.

The procedure OptimizeSet solves the QP restricted to $\mathcal{S}_i$ by sequential minimal optimization (Platt, 1998), in which we repeatedly select a pair of derivations d', d'' and optimize their dual variables $\alpha_{d'}, \alpha_{d''}$. The function SelectPair uses the heuristics suggested by Taskar (2004, p. 80) to select a pair of constraints: one must violate one of the KKT conditions ($\alpha_d(v_i(\mathbf{w}', d, d_i) - \xi_i = 0)$, and the other must allow the objective to be improved. The procedure OptimizePair optimizes a single pair of dual variables. This optimization is exact and can be derived as follows. Suppose we have current suboptimal weights $\mathbf{w}(\alpha) = \mathbf{w} + \eta \sum \alpha_d(\mathbf{h}(d_i) - \mathbf{h}(d))$, and we want to increase $\alpha_{d'}$ by δ and decrease $\alpha_{d''}$ by δ. Then we get the following optimization in a single variable, δ:

$$\begin{aligned} \text{maximize} \quad & -\frac{\eta}{2} \left\| \sum_d \alpha_d (\mathbf{h}(d_i) - \mathbf{h}(d)) + \delta(-\mathbf{h}(d') + \mathbf{h}(d'')) \right\|^2 + \delta(v_i(\mathbf{w}, d', d_i) - v_i(\mathbf{w}, d'', d_i)) \\ \text{subject to} \quad & -\alpha_{d'} \leq \delta \leq \alpha_{d''} \end{aligned} \tag{7}$$

Setting the partial derivative with respect to δ equal to zero, we get

$$\begin{aligned} \delta &= \frac{\eta \sum_d \alpha_d (\mathbf{h}(d_i) - \mathbf{h}(d)) \cdot (\mathbf{h}(d') - \mathbf{h}(d'')) + v_i(\mathbf{w}, d', d_i) - v_i(\mathbf{w}, d'', d_i)}{\eta \|\mathbf{h}(d') - \mathbf{h}(d'')\|^2} \\ &= \frac{(\mathbf{w}(\alpha) - \mathbf{w}) \cdot (\mathbf{h}(d') - \mathbf{h}(d'')) + v_i(\mathbf{w}, d', d_i) - v_i(\mathbf{w}, d'', d_i)}{\eta \|\mathbf{h}(d') - \mathbf{h}(d'')\|^2} \\ &= \frac{v_i(\mathbf{w}(\alpha), d', d_i) - v_i(\mathbf{w}(\alpha), d'', d_i)}{\eta \|\mathbf{h}(d') - \mathbf{h}(d'')\|^2} \end{aligned}$$

4. More accurately, we took the union of the 10 best derivations, the top 10 fear derivations, and the top 10 hope derivations (to be defined below).

Algorithm 2 MIRA weight update (Tsochantaridis et al., 2004; Platt, 1998; Taskar, 2004)

```
 1: procedure UpdateWeights($\mathbf{w}$, i)
 2:     $\epsilon = 0.01$
 3:     $\mathcal{S}_i \leftarrow \{d_i\}$
 4:     $again \leftarrow$ true
 5:     while $again$ do
 6:         $again \leftarrow$ false
 7:         $d^+ \leftarrow \arg\max_{d \in \mathcal{D}(f_i)} v_i(\mathbf{w}, d, d_i)$
 8:         if $v_i(\mathbf{w}, d^+, d_i) > \max_{d \in \mathcal{S}_i} v_i(\mathbf{w}, d, d_i) + \epsilon$ then
 9:             $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{d^+\}$
10:             OptimizeSet($\mathbf{w}, i$)
11:             $again \leftarrow$ true

12: procedure OptimizeSet($\mathbf{w}, i$)
13:     $\alpha_d \leftarrow 0$ for $d \in \mathcal{S}_i$
14:     $\alpha_{d_i} \leftarrow 1$
15:     $iterations \leftarrow 0$
16:     while $iterations < 1000$ do
17:         $iterations \leftarrow iterations + 1$
18:         $d', d'' \leftarrow$ SelectPair($\mathbf{w}, i$)
19:         if $d', d''$ not defined then
20:             return
21:         OptimizePair($\mathbf{w}, i, d', d''$)

22: function SelectPair($\mathbf{w}, i$)
23:     $\epsilon = 0.01$
24:     for $d' \in \mathcal{S}_i$ do
25:         $v_{max} \leftarrow \max_{d'' \neq d'} v_i(\mathbf{w}, d'', d_i)$
26:         if $\alpha_{d'} = 0$ and $v_i(\mathbf{w}, d', d_i) > v_{max} + \epsilon$ then
27:             if $\exists d'' \neq d'$ such that $\alpha_{d''} > 0$ then
28:                 return $d', d''$
29:         if $\alpha_{d'} > 0$ and $v_i(\mathbf{w}, d', d_i) < v_{max} - \epsilon$ then
30:             if $\exists d'' \neq d'$ such that $v_i(\mathbf{w}, d'', d_i) > v_i(\mathbf{w}, d', d_i)$ then
31:                 return $d', d''$
32:     return undefined

33: procedure OptimizePair($\mathbf{w}, i, d', d''$)
34:     $\delta \leftarrow \dfrac{v_i(\mathbf{w}, d', d_i) - v_i(\mathbf{w}, d'', d_i)}{\eta \|\mathbf{h}(d') - \mathbf{h}(d'')\|^2}$
35:     $\delta \leftarrow \max(-\alpha_{d'}, \min(\alpha_{d''}, \delta))$
36:     $\alpha_{d'} \leftarrow \alpha_{d'} + \delta$; $\alpha_{d''} \leftarrow \alpha_{d''} - \delta$
37:     $\mathbf{w} \leftarrow \mathbf{w} - \eta\delta(\mathbf{h}(d') - \mathbf{h}(d''))$
```

But in order to maintain constraint (7), we clip δ to the interval $[-\alpha_{d'}, \alpha_{d''}]$ (line 35).

At the end of training, following McDonald et al. (2005), we average all the weight vectors obtained at each iteration, just as in the averaged perceptron.

3.3 AROW

The conservativity term in (6) assumes that it is equally risky to move **w** in any direction, but this is not the case in general. For example, even a small change in the language model weights could result in a large change in translation length and fluency, whereas large changes in features like those attached to number-translation rules have a relatively small effect.

Imagine that we choose a feature of our model, h_j, and replace it with the feature $h_j \cdot c$ while replacing its weight with w_j/c. This change has no effect on the scores assigned to derivations or the translations generated, so intuitively one would hope that it also has no effect on learning. However, it is easy to see that our online algorithms in fact apply updates that are c times bigger, and relative to the new weight, c^2 times bigger.

A number of approaches are suggested in the literature to address this problem, for example, the second-order perceptron (Cesa-Bianchi et al., 2005), confidence-weighted learning (Dredze et al., 2008), and Adaptive Regularization of Weights or AROW (Crammer et al., 2009). AROW replaces the weight vector **w** with a Gaussian distribution over weight vectors, $\mathcal{N}(\mathbf{w}, \Sigma)$. The conservativity term in (6) accordingly changes from a Euclidean distance to a Kullback-Leibler distance. In addition, a new term is introduced that causes the confidence in the weights to increase over time (in AROW's predecessor (Dredze et al., 2008), it was motivated as the variance of L_i):

$$\mathbf{w}, \Sigma \leftarrow \arg\min_{\mathbf{w}', \Sigma'} \left(\mathrm{KL}\left(\mathcal{N}(\mathbf{w}', \Sigma') \parallel \mathcal{N}(\mathbf{w}, \Sigma)\right) + L_i(\mathbf{w}') + \frac{\lambda}{2} \mathbf{x}^{\mathsf{T}} \Sigma' \mathbf{x} \right)$$

In the original formulation of AROW for binary classification, **x** is the instance vector. Here, we set it to $\sum_{d \in \mathcal{S}_i} \alpha_d \left(\mathbf{h}(d_i) - \mathbf{h}(d)\right)$, even though the α_d aren't known in advance; in practice, they are known by the time they are needed.

With the KL distance between the two Gaussians written out explicitly, the quantity we want to minimize is

$$\frac{1}{2}\left(\log \frac{\det \Sigma}{\det \Sigma'} + \mathrm{Tr}\left(\Sigma^{-1}\Sigma'\right) + (\mathbf{w}' - \mathbf{w})^{\mathsf{T}} \Sigma^{-1} (\mathbf{w}' - \mathbf{w}) - D \right) + L_i(\mathbf{w}') + \frac{\lambda}{2} \mathbf{x}^{\mathsf{T}} \Sigma' \mathbf{x}$$

where D is the number of features. We minimize with respect to $\mathbf{w}'$ and Σ' separately. If we drop terms not depending on $\mathbf{w}'$, we get:

$$\mathbf{w} \leftarrow \arg\min_{\mathbf{w}'} \frac{1}{2} (\mathbf{w}' - \mathbf{w})^{\mathsf{T}} \Sigma^{-1} (\mathbf{w}' - \mathbf{w}) + L_i(\mathbf{w}')$$

which is the same as MIRA (6) except that Σ has taken the place of η. This leads to Algorithm 3, which modifies Algorithm 2 in two ways. First, line 34 is replaced with:

$$\delta \leftarrow \frac{v_i(\mathbf{w}, d', d_i) - v_i(\mathbf{w}, d'', d_i)}{(\mathbf{h}(d') - \mathbf{h}(d''))\Sigma(\mathbf{h}(d') - \mathbf{h}(d''))}$$

And line 37 is replaced with:

$$\mathbf{w} \leftarrow \mathbf{w} - \Sigma\delta(\mathbf{h}(d') - \mathbf{h}(d''))$$

Next, we turn to Σ. Setting partial derivatives with respect to Σ' to zero, and using the fact that Σ' is symmetric, we get (Petersen and Pedersen, 2008):

$$\frac{1}{2}\left(-\Sigma'^{-1} + \Sigma^{-1}\right) + \frac{\lambda}{2}\mathbf{x}^{\mathsf{T}}\mathbf{x} = 0$$

This leads to the AROW update, which follows the update for $\mathbf{w}$ (line 5 in Algorithm 1).

$$\Sigma^{-1} \leftarrow \Sigma^{-1} + \lambda \mathbf{x}^{\mathsf{T}}\mathbf{x}$$

We initialize Σ to $\eta_0 I$ and then update it at each iteration using this update; following Crammer et al. (2009), we keep only the diagonal elements of Σ.

Algorithm 3 AROW (Crammer et al., 2009)

```
Require: training examples $(f_1, e_1), \ldots, (f_N, e_N)$
 1: $\mathbf{w} \leftarrow \mathbf{0}$
 2: $\Sigma \leftarrow \eta_0 I$
 3: $\mathbf{s} \leftarrow \mathbf{0}, t \leftarrow 0$
 4: while not converged do
 5:     for $i \in \{1, \ldots, N\}$ in random order do
 6:         UpdateWeights($\mathbf{w}, i$)                    ▷ Algorithm 2
 7:         $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{w}$
 8:         $t \leftarrow t + 1$
 9:         $\mathbf{x} \leftarrow \sum_{d \in \mathcal{S}_i} \alpha_d (\mathbf{h}(d_i) - \mathbf{h}(d))$
10:         $\Sigma^{-1} \leftarrow \Sigma^{-1} + \lambda\, \mathrm{diag}(x_1^2, \ldots, x_n^2)$
11: $\mathbf{w} \leftarrow \mathbf{s}/t$

12: procedure OptimizePair($\mathbf{w}, i, d', d''$)
13:     $\delta \leftarrow \dfrac{v_i(\mathbf{w}, d', d_i) - v_i(\mathbf{w}, d'', d_i)}{(\mathbf{h}(d') - \mathbf{h}(d''))^{\mathsf{T}}\Sigma(\mathbf{h}(d') - \mathbf{h}(d''))}$
14:     $\delta \leftarrow \max(-\alpha_{d'}, \min(\alpha_{d''}, \delta))$
15:     $\alpha_{d'} \leftarrow \alpha_{d'} + \delta$
16:     $\alpha_{d''} \leftarrow \alpha_{d''} - \delta$
17:     $\mathbf{w} \leftarrow \mathbf{w} - \Sigma\delta(\mathbf{h}(d') - \mathbf{h}(d''))$
.
```

4. The reference derivation

We have been assuming that d_i is the derivation of the reference translation e_i. However, this is not always possible or even desirable. In this section, we discuss some alternative choices for d_i.

4.1 Bold/max-Bleu updating

It can happen that there does not exist any derivation of e_i, for example, if e_i contains a word never seen before in training. In this case, Liang et al. (2006), in the scheme they call *bold updating*, simply skip the sentence. Another approach, called *max*-Bleu *updating* (Tillmann and Zhang, 2006; Arun and Koehn, 2007), is to try to find the derivation with the highest Bleu score. However, Liang et al. find that even when it is possible to find a d_i that exactly generates e_i, it is not necessarily desirable to update the model towards it, because it may be a *bad derivation* of a *good translation*.

For example, consider the following Arabic sentence (written left-to-right in Buckwalter romanization) with English glosses:

(8)	sd	qTEp	mn	AlkEk	AlmmlH	“	brytzl	”	Hlqh
	blocked	piece	of	biscuit	salted	“	pretzel	”	his-throat

A very literal translation might be,

(9) A piece of a salted biscuit, a “pretzel,” blocked his throat

But the reference translation is in fact:

(10) A pretzel, a salted biscuit, became lodged in his throat

While still quite literal, translation (10) swaps grammatical roles in a way that is still difficult for statistical MT systems to model. If the system happens to have some bad rules that translate *sd qTEp mn* as *a pretzel* and *“ brytzl ”* as *became lodged in*, then it can use these bad rules to obtain a perfect translation, but using this derivation as the reference derivation would only reinforce the use of these bad rules. A derivation of translation (9) would probably serve better as the reference translation. What we need is a *good derivation* of a *good translation*.

4.2 Local updating

The most common way to do this has been to generate the n-best derivations according to the model and to choose the one with the lowest loss (Och and Ney, 2002). Liang et al. (2006) call this *local updating*. Watanabe et al. (2007) generate a 1000-best list and select either the derivation with lowest loss or the 10 derivations with lowest loss. The idea is that restricting to derivations with a higher model score will filter out derivations that use bad, low-probability rules. Normally one uses an n-best list as a proxy for the whole space of derivations, so that the larger n is, the better; in this case, however, as n increases, local updating approaches max-Bleu updating, which is what we are trying to avoid. It is not clear what the optimal n is, and whether it depends on factors such as sentence length or pruning.

4.3 Hope derivations

Here, we propose an approach that ties the choice of d_i more closely to the model. We suppose that for each f_i, the reference derivation d_i is unknown, and it doesn't necessarily derive the reference translation e_i, but we add a term to the objective function that says that we want d_i to have low loss relative to e_i.

$$\mathbf{w} \leftarrow \arg\min_{\mathbf{w}'} \min_{d_i \in \mathcal{D}(f_i)} \left(\frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}\|^2 + \max_{d \in \mathcal{D}(f_i)} v_i(\mathbf{w}', d, d_i) + (1-\mu)\ell_i(d_i, e_i) \right) \tag{11}$$

The parameter $\mu < 0$ controls how strongly we want d_i to have low loss.

We first optimize with respect to d_i, holding $\mathbf{w}'$ constant. Then the optimization reduces to

$$d_i = \arg\max_{d \in \mathcal{D}(f_i)} (\mu \ell_i(d, e_i) + \mathbf{w} \cdot \mathbf{h}(d)) \tag{12}$$

Then, we optimize with respect to $\mathbf{w}'$, holding d_i constant. Since this is identical to (6), we can use any of the algorithms presented in Section 3.

We call d_i chosen according to (12) the *hope* derivation. Unlike the fear derivation, it is parameterized by μ. If we let $\mu = -1$, the definition of the hope derivation becomes conveniently symmetric with the fear derivation:

$$d_i = \arg\max_{d \in \mathcal{D}(f_i)} (-\ell_i(d, e_i) + \mathbf{w} \cdot \mathbf{h}(d))$$

Both the hope and fear derivations try to maximize the model score, but the fear derivation maximizes the loss whereas the hope derivation minimizes the loss.

5. Searching for hope and fear

As mentioned above, one simple way of approximating either the hope or fear derivation is to generate an n-best list and choose from it the derivation that maximizes (12) or v_i, respectively. But Figure 1 shows that this approximation can be quite poor in practice, because the n-best list covers such a small portion of the entire search space. Increasing n would help (and, unlike with local updating, the larger n is, the better), but could become inefficient.

Instead, we use a dynamic program, analogous to the Viterbi algorithm, to directly search for the hope/fear derivations in the forest. (For efficiency, we reuse the forest that is previously used to search for the Viterbi derivation—an approximation, because this forest is pruned using the model score.) If our loss function were decomposable onto hyperedges, this would be a simple matter of setting the hyperedge weights to $\mathbf{w} \cdot \mathbf{h}(v \to \mathbf{v}) \pm \ell_i(v \to \mathbf{v})$ and running the Viterbi algorithm. However, our loss function is not hyperedge-decomposable, so we must resort to approximations.

5.1 Towards hyperedge-level BLEU

We begin by attempting to decompose the component scores $\mathbf{b}$ onto hyperedges. First, we need to be able to calculate $g_k(v \to \mathbf{v})$, the set of k-grams introduced by the hyperedge $(v \to \mathbf{v})$. This turns out to be fairly easy, because nearly all decoder implementations have a mechanism for scoring a k-gram language model, which is a feature of the form

$$h_{\mathrm{LM}_k}(d) = \sum_{w_1 \cdots w_k \in g_k(e(d))} \log P(w_k \mid w_1 \cdots w_{k-1}) \tag{13}$$

Since h_{LM_k} is decomposable onto hyperedges by assumption, it is safe to assume that g_k is also decomposable onto hyperedges, and so is n_k, which is the cardinality of g_k.

But m_k is not as easy to decompose, because of "clipping" of k-gram matches. Suppose our reference sentence is

(14) Australia is one of the few countries that have diplomatic relations with North Korea

and we have two partial translations

(15) the few

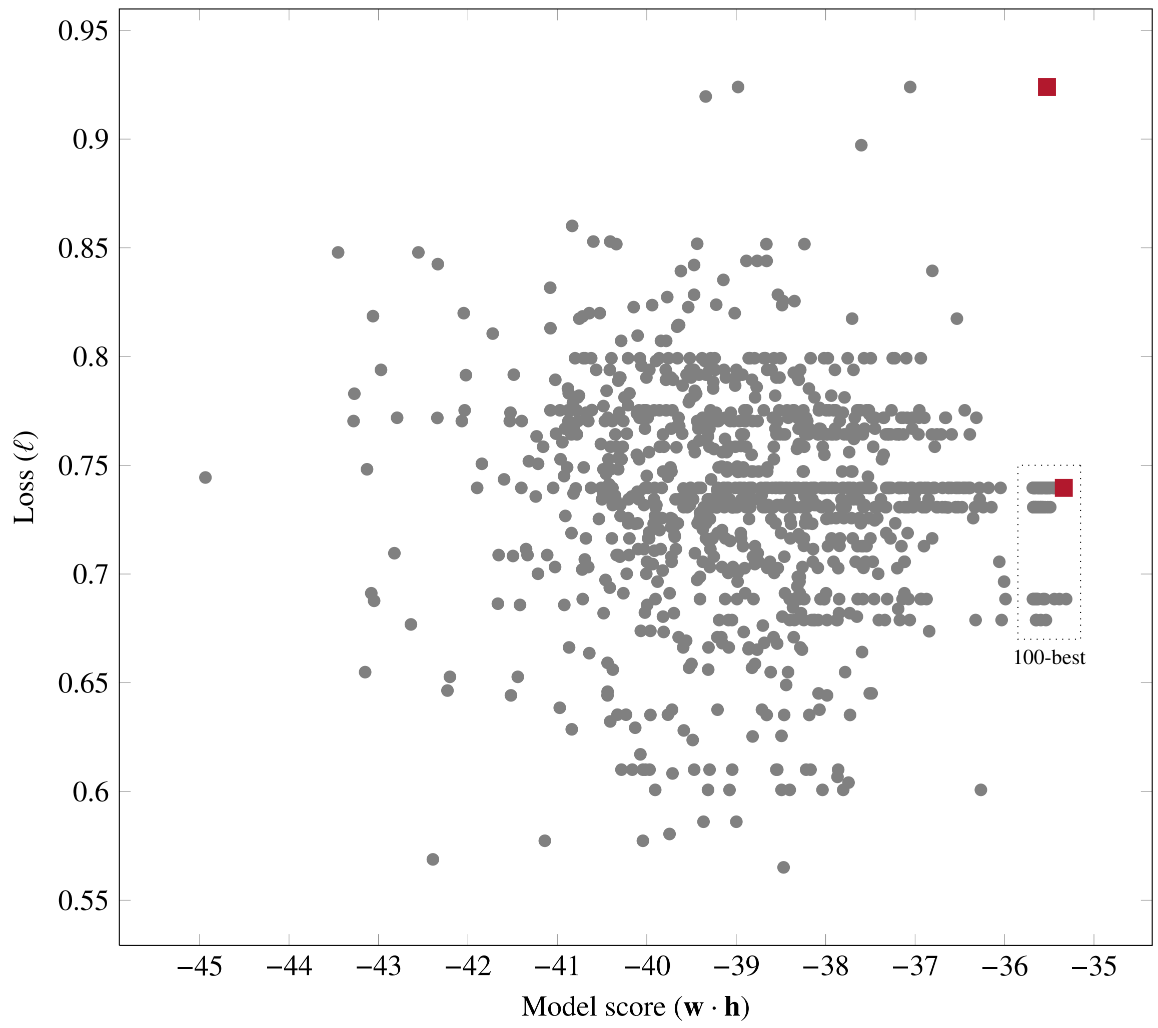

Figure 1: Using loss-augmented inference to search for fear translations in the whole forest is better than searching in the *n*-best list. Each point represents a derivation. The red square in the upper-right is the fear derivation obtained by loss-augmented inference, whereas the red square inside the box labeled "100-best" is the fear derivation selected from the 100-best list. (The gray circles outside the box are 100 random samples from the forest.)

(16) the countries

then for both, $m_1 = 2$. But if we combine them into

(17) the few the countries

then m_1 is not $2 + 2 = 4$, but 3, because *the* only occurs once in the reference sentence. In order to decompose m_k exactly, we would have to structure the forest hypergraph so that subderivations with different g_k are rooted at different nodes, resulting in an exponential blowup. Therefore, following Dreyer et al. (2007), we use *unclipped* counts of *n*-gram matches, which are not limited to the number of occurrences in the reference(s), in place of (1):

$$m_k = \left|g_k(c) \cap g_k(r)^*\right| \tag{18}$$

These counts are easily decomposable onto hyperedges.

Finally, in order to decompose ρ, if there are multiple references, we can't use the standard definition of ρ in (2); instead we use the average reference length. Then we can apportion ρ among hyperedges according to how much of the input sentence they consume:

$$\rho(v \to \mathbf{v}) = \frac{\rho}{|f_i|}\left(|f(v)| - \sum_{v' \in \mathbf{v}} \left|f(v')\right|\right) \tag{19}$$

where $f(v)$ is the part of the input sentence covered by the subderivation rooted at v.

5.2 Forest reranking

Appendix A.3, following Tromble et al. (2008), describes a way to fully decompose BLEU onto hyperedges. Here, however, we follow Dreyer et al. (2007), who use a special case of forest reranking (Huang, 2008). To search for the hope or fear derivation, we use the following dynamic program:

$$\begin{aligned} \mathit{vderiv}(v) &= \operatorname*{arg\,max}_{d \in \{\mathit{vderiv}(v \to \mathbf{v})\}} \phi(d) \\ \mathit{vderiv}(v \to \mathbf{v}) &= \{v \to \mathbf{v}\} \cup \bigcup_{v' \in \mathbf{v}} \mathit{vderiv}(v') \end{aligned}$$

where ϕ is one of the following:

$$\begin{aligned} \phi(d) &= \mathbf{w} \cdot \mathbf{h}(d) + B(\mathbf{b}(d, e_i)) && \text{hope} \\ \phi(d) &= \mathbf{w} \cdot \mathbf{h}(d) - B(\mathbf{b}(d, e_i)) && \text{fear} \end{aligned}$$

Note that maximizing $\mathbf{w} \cdot \mathbf{h}(d) + B(\mathbf{b}(d, e_i))$ is equivalent to maximizing $\mathbf{w} \cdot \mathbf{h}(d) - \ell_i(d, e_i)$, since they differ by only a constant; likewise, maximizing $\mathbf{w} \cdot \mathbf{h}(d) - B(\mathbf{b}(d, e_i))$ is equivalent to maximizing $\mathbf{w} \cdot \mathbf{h}(d) + \ell_i(d, e_i)$.

This algorithm is not guaranteed to find the optimum, however. We illustrate with a counterexample, using BLEU-2 (i.e., $K = 2$) instead of BLEU-4 for simplicity. Suppose our reference sentence is as above, and we have two partial candidate sentences

(20) one of the few nations which maintain ties with the DPRK has been

(21) North Korea with relations diplomatic have that countries few the of one is

Translation (20) has 4 unigram matches and 3 bigram matches, for a Bleu-2 score of $\sqrt{12/156}$; translation (21) has 13 unigram matches and 1 bigram match, for a Bleu-2 score of $\sqrt{13/156}$. If we extend both translations, however, with the word *Australia*, giving them each an extra unigram match, then translation (20) gets a Bleu-2 score of $\sqrt{15/156}$, and translation (21), $\sqrt{14/156}$. Though it does not always find the optimum, it works well enough in practice. After we find a hope or fear derivation, we recalculate its exact Bleu score, without any of the approximations described in this section.

6. Parallelization

Because inference is so slow for the translation task, and especially for the CKY-based decoder we are using, parallelization is critical. Batch learning algorithms like MERT are embarrassingly parallel, but parallelization of online learning is an active research area. Two general strategies have been proposed for SGD. The simpler strategy is to run p learners in parallel and then average their final weight vectors afterward (Mann et al., 2009; McDonald et al., 2010; Zinkevich et al., 2010). The more communication-intensive option, known as *asynchronous* SGD, is to maintain a single weight vector and for p parallel learners to update it simultaneously (Langford et al., 2009; Gimpel et al., 2010). It is not actually necessary for a learner to wait for the others to finish computing their updates; it can simply update the weight vector and move to the next example.

6.1 Iterative parameter mixing

A compromise between the two is *iterative parameter mixing* (McDonald et al., 2010), in which a master node periodically averages the weight vectors of the learners. At the beginning of each epoch, a master node broadcasts the same initial weight vector to p learners, which run in parallel over the training data and send their weight vectors back to the master node. The master averages the p weight vectors together to obtain the initial weight vector for the next epoch. At the end of training, the weight vectors from each iteration of each learner are all averaged together to yield the final weight vector.

6.2 Asynchronous MIRA/AROW

In asynchronous SGD, when multiple learners make simultaneous updates to the master weight vector, the updates are simply summed. Our experience is that this works, but requires carefully throttling back the learning rate η. Here, we focus on asynchronous parallelization of MIRA/AROW. The basic idea is to build forests for several examples in parallel, and optimize the QP over all of them together. However, this would require keeping the forests of all the examples in a shared memory, which would probably be too expensive. Instead, the solution we have adopted (Algorithm 4) is for the learners to broadcast just the working sets $\mathcal{S}_i$ to one another, rather than whole forests. Thus, when each learner works on a training example (f_i, e_i), it optimizes the QP on it along with all of the working sets it received from other nodes. It can grow the working set $\mathcal{S}_i$, but not the working sets it received from other nodes. For AROW, each node maintains its own Σ in addition to its own $\mathbf{w}$.

Algorithm 4 Asynchronous MIRA

```
 1: $\mathbf{w}_k \leftarrow \mathbf{0}$ for each node $k$
 2: $\mathbf{s}_k \leftarrow \mathbf{0}, t_k \leftarrow 0$ for each node $k$
 3: while not converged do
 4:     $T \leftarrow$ training data
 5:     for each node $k$ in parallel do
 6:         while $T \neq \emptyset$ do
 7:             pick a random $(f_i, e_i)$ from $T$ and remove it
 8:             receive working sets $\{\mathcal{S}_{i'} \mid i' \in I\}$ from other nodes
 9:             UpdateWeights($\mathbf{w}_k, i, I$)
10:             broadcast $\mathcal{S}_i$ to other nodes
11:             $\mathbf{s}_k \leftarrow \mathbf{s}_k + \mathbf{w}_k$
12:             $t_k \leftarrow t_k + 1$
13: $\mathbf{w} \leftarrow \dfrac{\sum_k \mathbf{s}_k}{\sum_k t_k}$

14: procedure UpdateWeights($\mathbf{w}, i, I$)
15:     $\epsilon = 0.01$
16:     $\mathcal{S}_i \leftarrow \{d_i\}$
17:     again $\leftarrow$ true
18:     while again do
19:         again $\leftarrow$ false
20:         $d^+ \leftarrow \arg\max_{d \in \mathcal{D}(f_i)} v_i(\mathbf{w}, d, d_i)$
21:         if $v_i(\mathbf{w}, d^+, d_i) > \max_{d \in \mathcal{S}_i} v_i(\mathbf{w}, d, d_i) + \epsilon$ then
22:             $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{d^+\}$
23:             again $\leftarrow$ true
24:         if again then
25:             OptimizeSets($\mathbf{w}, \{i\} \cup I$)

26: procedure OptimizeSets($\mathbf{w}, I$)
27:     for $i \in I$ do
28:         $\alpha_d \leftarrow 0$ for $d \in \mathcal{S}_i$
29:         $\alpha_{d_i} \leftarrow 1$
30:     again $\leftarrow$ true
31:     iterations $\leftarrow 0$
32:     while again and iterations $< 1000$ do
33:         again $\leftarrow$ false
34:         iterations $\leftarrow$ iterations $+ 1$
35:         for $i \in I$ do
36:             $d', d'' \leftarrow$ SelectPair($\mathbf{w}, i$)                ▷ Algorithm 2
37:             if $d', d''$ defined then
38:                 OptimizePair($\mathbf{w}, i, d', d''$)
39:                 again $\leftarrow$ true
```

7. Experiments

We experimented with the methods described above on the hierarchical phrase-based translation system Hiero (Chiang, 2005, 2007), using two feature sets. The *small* model comprises 13 features: 7 inherited from Pharaoh (Koehn et al., 2003), a second language model, and penalties for the glue rule, identity rules, unknown-word rules, and two kinds of number/name rules. The *large* model additionally includes the following lexical features:

- $\text{lex}(e)$ fires when an output word e is generated
- $\text{lex}(f, e)$ fires when an output word e is generated aligned to a input word f
- $\text{lex}(\text{NULL}, e)$ fires when an output word e is generated unaligned

In all these features, f and e are limited to words occurring 10,000 times or more in the parallel data; less-frequent words are replaced with the special symbol UNK. Typically, this results in 10,000–20,000 features.

Our training data were all drawn from the constrained track of the NIST 2009 Open Machine Translation Evaluation. We extracted an Arabic-English grammar from all the allowed parallel data (152+175M words), and we trained two 5-gram language models, one on the combined English sides of the Arabic-English and Chinese-English tracks (385M words), and another on 2 billion words of English.

We ran discriminative training on 3011 lines (67k Arabic words) of newswire and web data drawn from the NIST 2004 and 2006 evaluations and newsgroup data from the GALE program (LDC2006E92). After each epoch (pass through the discriminative-training data), we used the averaged weights to decode our development data, which was from the NIST 2008 evaluation (1357 lines, 36k Arabic words). After 10 epochs, we chose the weights that yielded the highest BLEU on the development data and decoded the test data, which was from the NIST 2009 evaluation (1313 lines, 34k Arabic words).

Except where noted, the following default settings were used:

- Learning rate $\eta = 0.01$
- Hope derivations with $\mu = -1$
- Forest reranking for hope/fear derivations
- Iterative parameter mixing on 20 processors

A few probability features have to be initialized carefully: the two language models and the two phrase translation probability models. If these features are given negative weights, extremely long and disfluent translations result, and we find that the learner has difficulty recovering. So we initialize their weights to 1 instead of 0, and in AROW, we initialize their learning rates to 0.01 instead of η_0.

The learning curves in the figures referenced below show the BLEU score obtained on the development data (disjoint from the discriminative-training data) over time. Figure 2abc shows learning curves for SGD, MIRA, and minimum risk (see Appendix A) for several values of the learning rate η, using the small model. Generally, all the methods converged to the same performance level, and SGD and minimum risk were surprisingly not very sensitive to the learning rate η. MIRA, on the

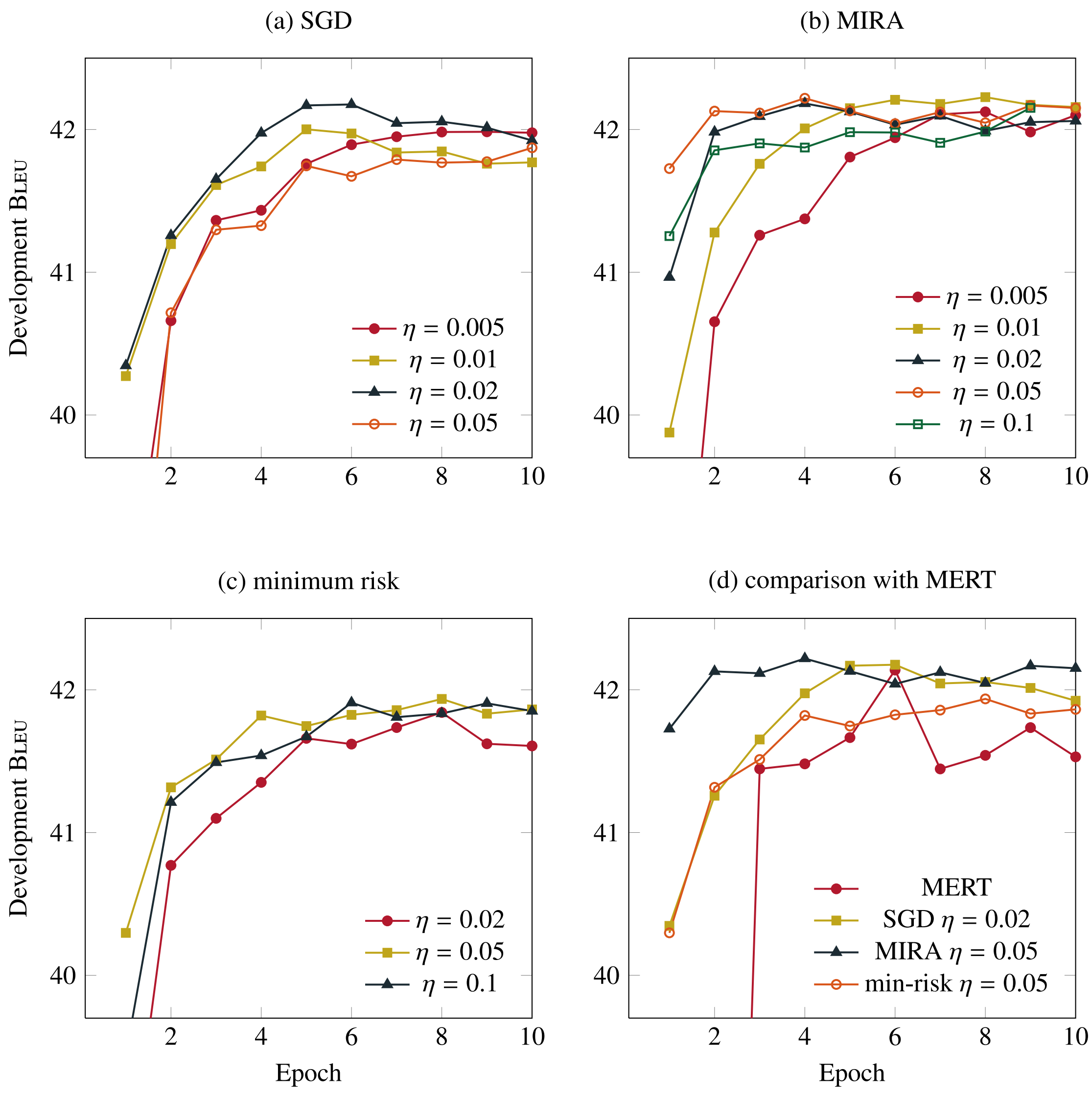

Figure 2: Learning curves of various algorithms on the development data, using the small model. Graphs (a), (b), and (c) show the effect of the learning rate η on SGD, MIRA, and minimum risk. SGD and min-risk seem relatively insensitive to η, while MIRA converges faster with higher η. Graph (d) compares the three online methods against MERT. The online algorithms converge more quickly and smoothly than MERT does, with MIRA slightly better than the others. The first two epochs of MERT, not shown here, had scores of 10.6 and 31.6.

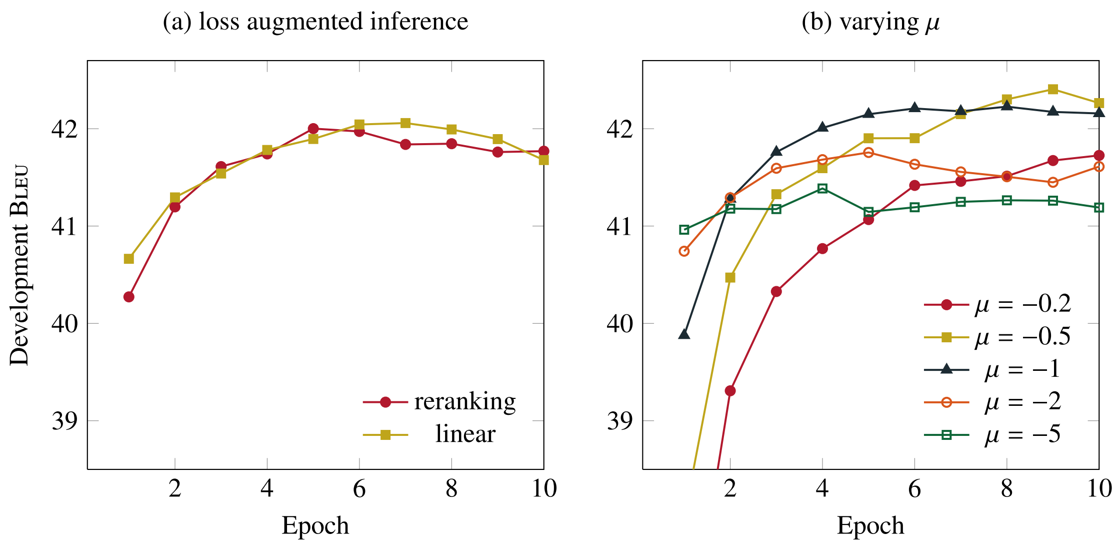

Figure 3: Variations on selecting hope/fear derivations, using the small model. (a) Linear BLEU performs as well as or slightly better than forest reranking. SGD, $\eta = 0.01$. (b) More negative values of the loss weight μ for hope derivations lead to higher initial performance, whereas less negative loss weights lead to higher final performance. MIRA, $\eta = 0.01$.

other hand, converged faster with higher learning rates up to $\eta = 0.05$. Since our past experience suggests that on tasks with lower BLEU scores (namely, Chinese-English web and speech), lower learning rates are better, our default $\eta = 0.01$ seems like a generally safe value.

Figure 2d compares all three algorithms with MERT (20 random restarts). The online algorithms converge more quickly and smoothly than MERT does, with MIRA converging faster than the others. However, on the test set (Table 1), MERT outperformed the other algorithms. Using bootstrap resampling with 1000 samples (Koehn, 2004; Zhang et al., 2004), only the difference with minimum risk was significant ($p < 0.05$).

One possible confounding factor in our comparison with minimum risk is that it must use linear BLEU to compute the gradient. To control for this, we ran SGD (on the hinge loss) using both forest reranking and linear BLEU to search for hope/fear derivations (Figure 3a). We found that their performance is quite close, strengthening our finding that the hinge loss performs slightly better than minimum risk.

Figure 3b compares several values of the parameter μ that controls how heavily to weight the loss function when computing hope derivations. Higher loss weights lead to higher initial performance, whereas lower loss weights lead to higher final performance (the exception being $\mu- = 0.2$, which perhaps would have improved with more time). A weight of $\mu = -1$ appears to be a good tradeoff, and is symmetrical with the weight of 1 used when computing fear derivations. It would be interesting, however, to investigate decaying the loss weight over time, as proposed by McAllester et al. (2010).

We then compared the two methods of parallelization (Figure 4). These experiments were run on a cluster of nodes communicating by MPI (Message Passing Interface) over Myrinet, a high-speed

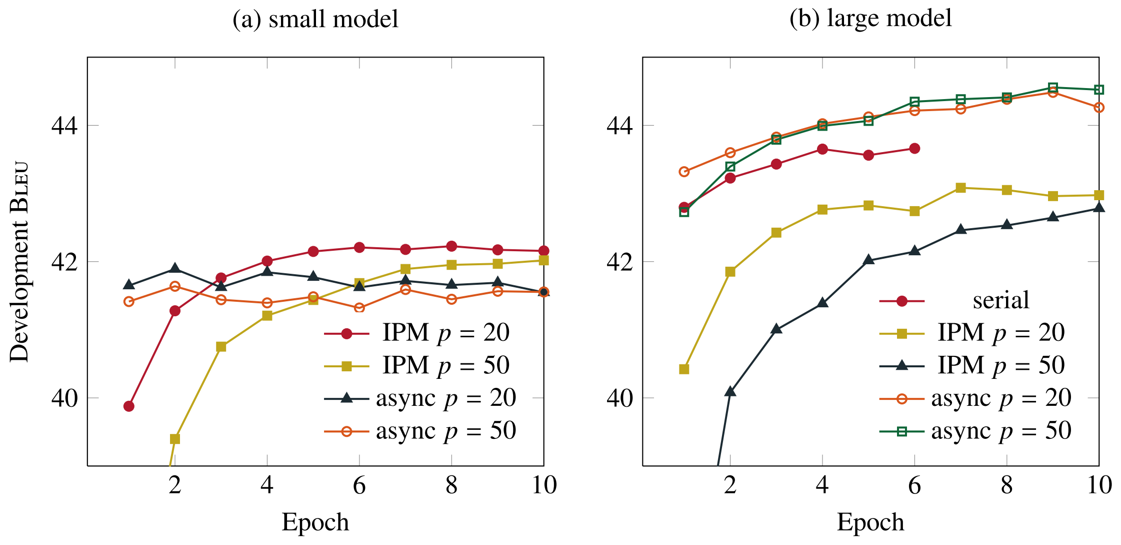

Figure 4: On the small model, asynchronous MIRA does not perform well compared to iterative parameter mixing. But on the large model, asynchronous MIRA strongly outperforms iterative parameter mixing. Increasing the number of processors to 50 provides little benefit to iterative parameter mixing in either case, whereas asynchronous MIRA gets a near-linear speedup.

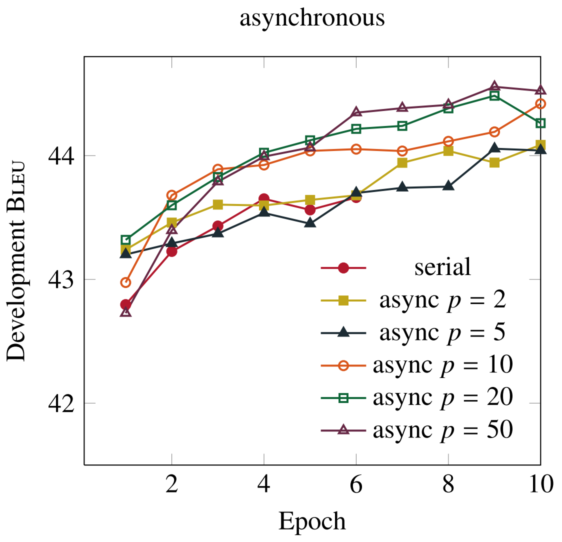

Figure 5: Taking a closer look at asynchronous sharing of working sets, we see that, at each epoch, greater parallelization generally gives better performance.

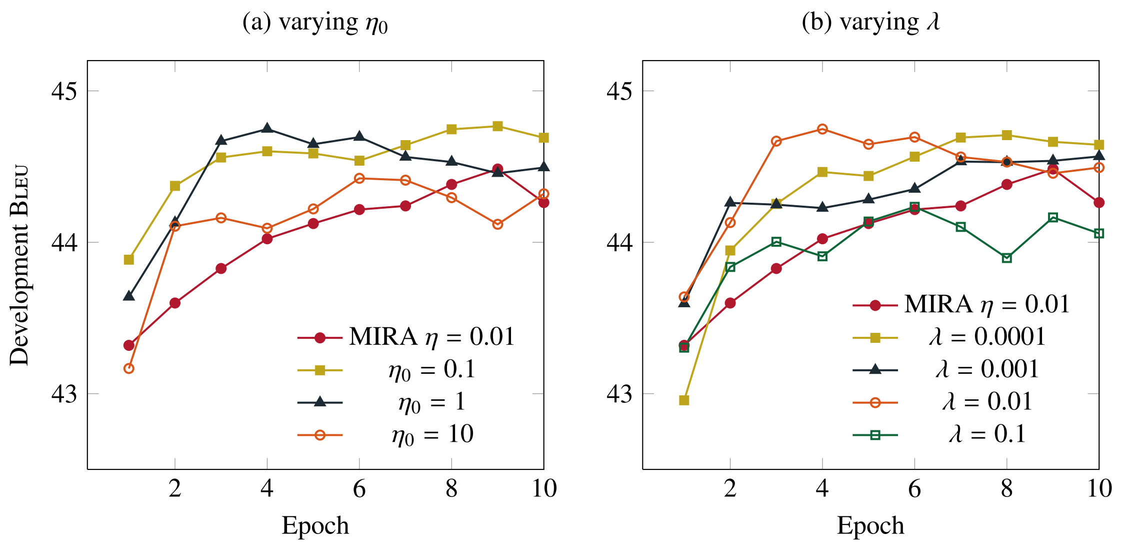

Figure 6: (a) With $\lambda = 0.01$, AROW seems relatively insensitive to the choice of η_0 in the range of 0.1 to 1, but performs much worse outside that range. (b) With $\eta_0 = 1$, AROW converges faster for larger values of λ up to 0.01; at 0.1, however, the algorithm appears to be unable to make progress.

local area networking system. In these graphs, the x-axis continues to be the number of epochs; wallclock time is roughly proportional to the number of epochs divided by p, but mixed hardware unfortunately prevented us from performing direct comparisons of wallclock time.

One might expect that, at each epoch, the curves with greater p underperform the curves with lower p only slightly. With iterative parameter mixing, for both the small and large models, we see that increasing p from 20 to 50 degrades performance considerably. It would appear that there is very little speedup due to parallelization, probably because the training data is so small (3011 sentences).

Asynchronous MIRA using the small model starts off well but afterwards does not do as well as iterative parameter mixing. On the large model, however, asynchronous MIRA performs dramatically better. Taking a closer look at its performance for varying p (Figure 5), we see that, at each epoch, the curves with greater p actually tend to outperform the curves with lower p.

Next, we tested the AROW algorithm. We held λ fixed to 0.01 and compared different values of the initial learning rate η_0 (Figure 6a, finding that the algorithm performed well for $\eta_0 = 0.1$ and 1 and was fairly insensitive to the choice of η_0 in that range; larger and smaller values, however, performed worse. We then held $\eta_0 = 1$ and compared different values of λ (Figure 6b), finding that higher values converged faster, but $\lambda = 0.1$ did much worse.

The scores on the test set (Table 1) using the large model generally confirm what was already observed on the development set. In total, the improvement over MERT on the test set is 2.4 Bleu.

model	obj	alg	approx	par	epoch	BLEU dev	BLEU test
small	1 − BLEU	MERT	–	–	6	42.1	45.2
small	hinge	SGD $\eta = 0.02$	rerank	IPM	6	42.2	44.9
small	risk	SGD $\eta = 0.05$	linear	IPM	8	41.9	44.8
small	hinge	MIRA $\eta = 0.05$	rerank	IPM	4	42.2	44.9
large	hinge	SGD $\eta = 0.01$	rerank	IPM	5	42.4	45.2
large	hinge	MIRA $\eta = 0.01$	rerank	IPM	7	43.1	45.9
large	hinge	MIRA $\eta = 0.01$	rerank	async	9	44.5	47.3
large	hinge	AROW $\eta_0 = 1$ $\lambda = 0.01$	rerank	async	4	44.7	47.6

Table 1: Final results. Key to columns: **model** = features used, **obj** = objective function, **alg** optimization algorithm, **approx** = approximation for calculating the loss function on forests, **par** = parallelization method, **epoch** = which epoch was selected on the development data, **dev** and **test** = (case-insensitive IBM) BLEU score on development and test data (NIST 2008 and 2009, respectively).

8. Conclusion

We have surveyed several methods for online discriminative training and the issues that arise in adapting these methods to the task of statistical machine translation. Using SGD, we found that the large-margin objective performs slightly better than minimum risk. Then, using the large-margin objective, we found that MIRA does better than SGD, and AROW, better still. We extended all of these methods in novel ways to cope with the large structured search space of the translation task, that is, to use as much of the translation forest as possible.

An apparent disadvantage of the large-margin objective is its requirement of a single correct derivation, which does not exist. We showed that the *hope* derivation serves this purpose well. We demonstrated that the highest-BLEU derivation is not in general the right choice, by showing that performance drops for very negative values of μ. We also raised the possibility, as yet unexplored, of decaying μ over time, as has been suggested by McAllester et al. (2010).

The non-decomposability of BLEU as a loss function is a nuisance that must be dealt with carefully. However, the choice of approximation (forest reranking versus linear BLEU) for loss-augmented inference or expectations turned out not to be very important. Past experience shows that linear BLEU sometimes outperforms and sometimes underperforms forest reranking, but since it is faster and easier to implement, it may be the better choice.

The choice of parallelization method turned out to be critical. We found that asynchronous sharing of working sets in MIRA/AROW not only gave speedups that were nearly linear in the number of processors, but also gave dramatically higher final BLEU scores than iterative parameter mixing. It is not clear yet whether this is because iterative parameter mixing was not able to converge in only 10 epochs or because aggregating working sets confers an additional advantage.

Although switching from MERT to online learning initially hurt performance, by adding some very simple features to the model, we ended up with a gain of 2.4 BLEU over MERT. When these online methods are implemented with due attention to translation forests, the nature of the translation problem, the idiosyncrasies of BLEU, and parallelization, they are a highly effective vehicle for exploring new extensions to discriminative models for translation.

Acknowledgements

This work evolved over time to support several projects adding new features to the ISI machine translation systems, and would not have been possible without my collaborators on those projects: Steve DeNeefe, Kevin Knight, Yuval Marton, Michael Pust, Philip Resnik, and Wei Wang. I also thank Michael Bloodgood, Michael Collins, John DeNero, Vladimir Eidelman, Kevin Gimpel, Chun-Nan Hsu, Daniel Marcu, Ryan McDonald, Fernando Pereira, Fei Sha, and the anonymous reviewers for their valuable ideas and feedback. This work was supported in part by DARPA under contract HR0011-06-C-0022 (subcontract to BBN Technologies), HR0011-09-1-0028, and DOI-NBC D11AP00244. *S.D.G.*

Appendix A. Minimum risk training

In this appendix, we describe minimum risk (expected loss) training (Smith and Eisner, 2006; Zens et al., 2008; Li and Eisner, 2009; Arun et al., 2010) and some notes on its implementation.

A.1 Objective function

Define a probabilistic version of the model,

$$P_T(d \mid f_i) \propto \exp \frac{1}{T} \mathbf{w} \cdot \mathbf{h}(d)$$

where T is a temperature parameter, and for any random variable X over derivations, define

$$E_T[X \mid f_i] = \sum_{d \in \mathcal{D}(f_i)} P_T(d \mid f_i) X(d)$$

In minimum-risk training, we want to minimize $\sum_i E_T[\ell_i(d, d_i) \mid f_i]$ for $T = 1$. In annealed minimum-risk training (Smith and Eisner, 2006), we let $T \to 0$, in which case the expected loss approaches the loss.

This objective function is differentiable everywhere (unlike in MERT), though not convex (as maximum likelihood is). The gradient for a single example is:

$$\nabla E_T[\ell_i(d, d_i) \mid f_i] = \frac{1}{T} \left(E_T[\ell_i \mathbf{h} \mid f_i] - E_T[\ell_i \mid f_i] E_T[\mathbf{h} \mid f_i] \right)$$

or, in terms of B:

$$\begin{aligned} \nabla E_T[\ell_i(d, d_i) \mid f_i] &= -\nabla E_T[B(\mathbf{b}(d, e_i)) \mid f_i] \\ &= -\frac{1}{T} \left(E_T[B\mathbf{h} \mid f_i] - E_T[B \mid f_i] E_T[\mathbf{h} \mid f_i] \right) \end{aligned} \tag{22}$$

A major advantage that minimum-risk has over the large-margin methods explored in this paper is that it does not require a reference derivation, or a hope derivation as a proxy for the reference derivation. The main challenge with minimum-risk training is that we must calculate expectations of B and $B\mathbf{h}$. We discuss how this is done below.

A.2 Relationship to hope/fear derivations

There is an interesting connection between the risk and the generalized hinge loss (4). McAllester et al. (2010) show that for applications where the input space is continuous (as in speech processing), a perceptron-like update using the hope and 1-best derivations, or the 1-best and fear derivations, approaches the gradient of the loss. We provide here an analogous argument for the discrete input case.

Consider a single training example (f_i, e_i), so that we can simply write ℓ for ℓ_i and $E_T[X]$ for $E_T[X \mid f_i]$. Define a loss-augmented model:

$$P_\mu(d \mid f_i) \propto \exp \frac{1}{\mu} (\mathbf{w} \cdot \mathbf{h}(d) + \mu\ell(d, d_i))$$

and define

$$E_\mu[X] = \sum_{d \in \mathcal{D}(f_i)} P_\mu(d \mid f_i) X(d)$$

As before, the gradient with respect to $\mathbf{w}$ is:

$$\nabla_{\mathbf{w}} E_\mu[\ell] = \frac{1}{T} \Big(E_\mu[\ell \mathbf{h}] - E_\mu[\ell] E_\mu[\mathbf{h}] \Big)$$

and, by the same reasoning, the partial derivative of $E[\mathbf{h}]$ with respect to μ comes out to be the same:

$$\frac{\partial}{\partial \mu} E_\mu[\mathbf{h}] = \frac{1}{T} \Big(E_\mu[\mathbf{h}\ell] - E_\mu[\mathbf{h}] E_\mu[\ell] \Big)$$

Therefore we have

$$\begin{aligned} \nabla_{\mathbf{w}} E[\ell] &= \left. \frac{\partial E_\mu[\mathbf{h}]}{\partial \mu} \right|_{\mu=0} \\ &= \lim_{\mu \to 0} \frac{1}{2\mu} \Big(E_\mu[\mathbf{h}] - E_{-\mu}[\mathbf{h}] \Big) \end{aligned}$$

which suggests the following update rule:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta'}{2\mu} \Big(E_\mu[\mathbf{h}] - E_{-\mu}[\mathbf{h}] \Big)$$

with μ decaying over time. But if we let $\mu = 1$ (that is, to approximate the tangent with a secant), and $\eta' = 2\eta$, we get:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \, (E_{+1}[\mathbf{h}] - E_{-1}[\mathbf{h}])$$

Having made this approximation, there is no harm in letting $T = 0$, so that the expectations of $\mathbf{h}$ become the value of $\mathbf{h}$ at the mode of the underlying distribution:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \, (\mathbf{h}(d_{+1}) - \mathbf{h}(d_{-1})) \\ d_{+1} &= \arg\max_d \, (\mathbf{w} \cdot \mathbf{h}(d) + \ell(d, d_i)) \\ d_{-1} &= \arg\max_d \, (\mathbf{w} \cdot \mathbf{h}(d) - \ell(d, d_i)) \end{aligned}$$

But this is exactly the SGD update on the generalized hinge loss (5), with $d^+ = d_{+1}$ being the fear derivation and $d_i = d_{-1}$ being the hope derivation.

A.3 Linear Bleu

In order to calculate the expected loss from a forest of derivations, we must make the loss fully decomposable onto hyperedges. Tromble et al. (2008) define a linear approximation to Bleu which they use for minimum Bayes risk decoding. We present here a version that includes the brevity penalty.

Suppose we have some fixed document with component scores $\bar{\mathbf{b}}$ and add a sentence to it that has component scores $\mathbf{b}$. How does adding the new sentence affect the Bleu score? Form a first-order Taylor approximation around $\bar{\mathbf{b}}$:

$$\begin{aligned}\textsc{Bleu}(\bar{\mathbf{b}} + \mathbf{b}) &\approx \textsc{Bleu}(\bar{\mathbf{b}}) + \mathbf{b} \cdot \nabla\textsc{Bleu}(\bar{\mathbf{b}}) \\ &= \textsc{Bleu}(\bar{\mathbf{b}}) \left(1 + \sum_{k=1}^{K} \left(\frac{m_k}{K\bar{m}_k} - \frac{n_k}{K\bar{n}_k}\right) + H(\bar{\rho} - \bar{n}_1)\left(\frac{\bar{\rho} n_1}{\bar{n}_1^2} - \frac{\rho}{\bar{n}_1}\right)\right)\end{aligned}$$

where

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Note that although the brevity penalty is not differentiable at $\bar{n}_1 = \bar{\rho}$, we have filled in an arbitrary value (which is easier than smoothing the brevity penalty and works well in practice).

Since this approximation is linear in the m_k and n_k, it is decomposable onto hyperedges. The term involving ρ is the same for all derivations, so we don't need to decompose it and can also skip (19).

The approximation is highly dependent on $\bar{\mathbf{b}}$; Tromble et al. use a fixed $\bar{\mathbf{b}}$ but we use the oracle document defined in Section 2.4. Then B, defined as in (3) but using the linear approximation to Bleu, is decomposable down to hyperedges, making it possible to compute $E[B]$ as well as $E[b\mathbf{h}]$ over the entire forest.

A.4 Calculating the risk and its gradient

To calculate the expected loss, we can use the expectation semiring of Eisner (2002); we give a slightly modified definition that renormalizes intermediate values in such a way that they can be stored directly instead of as signed logarithms:

$$\mathit{expect}_B(v) = \sum_{(v \to \mathbf{v}) \in E} \frac{\mathit{inside}_{\mathbf{w}\cdot\mathbf{h}}(v \to \mathbf{v})}{\mathit{inside}_{\mathbf{w}\cdot\mathbf{h}}(v)} \mathit{expect}_B(v \to \mathbf{v}) \tag{23}$$

$$\mathit{expect}_B(v \to \mathbf{v}) = B(v \to \mathbf{v}) + \sum_{v' \in \mathbf{v}} \mathit{expect}_B(v') \tag{24}$$

$$\mathit{inside}_{\mathbf{w}\cdot\mathbf{h}}(v) = \sum_{(v \to \mathbf{v}) \in E} \mathit{inside}_{\mathbf{w}\cdot\mathbf{h}}(v \to \mathbf{v})$$

$$\mathit{inside}_{\mathbf{w}\cdot\mathbf{h}}(v \to \mathbf{v}) = \exp \mathbf{w} \cdot \mathbf{h}(v \to \mathbf{v}) \times \prod_{v' \in \mathbf{v}} \mathit{inside}_{\mathbf{w}\cdot\mathbf{h}}(v')$$

To calculate the expected product $E_T[B\mathbf{h} \mid f_i]$ in the gradient (22), we use the second-order expectation semiring (Li and Eisner, 2009), similarly modified:

$$\begin{aligned}
expect_{B\mathbf{h}}(v) &= \sum_{(v\to\mathbf{v})\in E} \frac{inside_{\mathbf{w}\cdot\mathbf{h}}(v \to \mathbf{v})}{inside_{\mathbf{w}\cdot\mathbf{h}}(v)} expect_{B\mathbf{h}}(v \to \mathbf{v}) \\
expect_{B\mathbf{h}}(v \to \mathbf{v}) &= expect_B(v \to \mathbf{v}) expect_{\mathbf{h}}(v \to \mathbf{v}) \\
&\quad + \sum_{v'\in\mathbf{v}} (expect_{B\mathbf{h}}(v') - expect_B(v') expect_{\mathbf{h}}(v'))
\end{aligned}$$

where $expect_{\mathbf{h}}$ is calculated analogously to $expect_B$ (23–24).

References

Abhishek Arun and Philipp Koehn. Online learning methods for discriminative training of phrase based statistical machine translation. In *Proceedings of MT Summit XI*, 2007.

Abhishek Arun, Barry Haddow, and Philipp Koehn. A unified approach to minimum risk training and decoding. In *Proceedings of the Fifth Workshop on Statistical Machine Translation*, 2010.

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

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

Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. A second-order perceptron algorithm. *SIAM Journal on Computing*, 34(3):640–668, 2005.

David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of ACL*, 2005.

David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2), 2007.

David Chiang. Learning to translate with source and target syntax. In *Proceedings of ACL*, 2010.

David Chiang, Steve DeNeefe, Yee Seng Chan, and Hwee Tou Ng. Decomposability of translation metrics for improved evaluation and efficient algorithms. In *Proceedings of EMNLP*, 2008a.

David Chiang, Yuval Marton, and Philip Resnik. Online large-margin training of syntactic and structural translation features. In *Proceedings of EMNLP*, 2008b.

David Chiang, Wei Wang, and Kevin Knight. 11,001 new features for statistical machine translation. In *Proceedings of NAACL HLT*, 2009.

David Chiang, Steve DeNeefe, and Michael Pust. Two easy improvements to lexical weighting. In *Proceedings of ACL HLT*, 2011.

Michael Collins. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, 2002.

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

Koby Crammer, Alex Kulesza, and Mark Dredze. Adaptive regularization of weight vectors. In *Advances in Neural Information Processing Systems 22*, 2009.

Harold Charles Daumé, III. *Practical Structured Learning Techniques for Natural Language Processing*. PhD thesis, University of Southern California, 2006.

Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of ICML*, 2008.

Markus Dreyer, Keith Hall, and Sanjeev Khudanpur. Comparing reordering constraints for SMT using efficient BLEU oracle computation. In *Proceedings of the Workshop on Syntax and Structure in Statistical Translation*, 2007.

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of ACL*, 2002.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1999.

Kevin Gimpel, Dipanjan Das, and Noah A. Smith. Distributed asynchronous online learning for natural language processing. In *Proceedings of CoNLL*, 2010.

Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL*, 2008.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–64, 1996.

Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of EMNLP*, 2004.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of HLT-NAACL*, 2003.

John Langford, Alexander J. Smola, and Martin Zinkevich. Slow learners are fast. In *Advances in Neural Information Processing Systems 22*, 2009.

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

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

Chin-Yew Lin and Franz Josef Och. ORANGE: A method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of COLING*, 2004.

William N. Locke and A. Donald Booth, editors. *Machine Translation of Languages: Fourteen Essays*. Technology Press of MIT, Cambridge, MA, 1955.

Gideon Mann, Ryan McDonald, Mehryar Mohri, Nathan Silberman, and Daniel D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems 22*, 2009.

David McAllester, Tamir Hazan, and Joseph Keshet. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems 23*, 2010.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of ACL*, 2005.

Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Proceedings of NAACL HLT*, 2010.

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

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

Franz Josef Och, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proceedings of EMNLP*, 1999.

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

Kaare Brandt Petersen and Michael Syskind Pedersen. *The Matrix Cookbook*. 2008. `http://matrixcookbook.com`.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 195–208. MIT Press, 1998.

Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Subgradient methods for maximum margin structured learning. In *Proceedings of the ICML Workshop on Learning in Structured Output Spaces*, 2006.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of ICML*, 2007.

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

Ben Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University, 2004.

Christoph Tillmann and Tong Zhang. A discriminative global training algorithm for statistical MT. In *Proceedings of COLING-ACL*, 2006.

Roy W. Tromble, Shankar Kumar, Franz Och, and Wolfgang Macherey. Lattice minimum Bayes-risk decoding for statistical machine translation. In *Proceedings of EMNLP*, 2008.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of ICML*, 2004.

Taro Watanabe, Jun Suzuki, Hajime Tsukuda, and Hideki Isozaki. Online large-margin training for statistical machine translation. In *Proceedings of EMNLP*, 2007.

Richard Zens, Saša Hasan, and Hermann Ney. A systematic comparison of training criteria for statistical machine translation. In *Proceedings of EMNLP*, 2008.

Ying Zhang, Stephan Vogel, and Alex Waibel. Interpreting BLEU/NIST scores: How much improvement do we need to have a better system? In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC)*, 2004.

Martin A. Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23*, 2010.