Tuning as Ranking

Mark Hopkins and Jonathan May
SDL Language Weaver
Los Angeles, CA 90045
{mhopkins,jmay}@sdl.com

Abstract

We offer a simple, effective, and scalable method for statistical machine translation parameter tuning based on the pairwise approach to ranking (Herbrich et al., 1999). Unlike the popular MERT algorithm (Och, 2003), our pairwise ranking optimization (PRO) method is not limited to a handful of parameters and can easily handle systems with thousands of features. Moreover, unlike recent approaches built upon the MIRA algorithm of Crammer and Singer (2003) (Watanabe et al., 2007; Chiang et al., 2008b), PRO is easy to implement. It uses off-the-shelf linear binary classifier software and can be built on top of an existing MERT framework in a matter of hours. We establish PRO's scalability and effectiveness by comparing it to MERT and MIRA and demonstrate parity on both phrase-based and syntax-based systems in a variety of language pairs, using large scale data scenarios.

1 Introduction

The MERT algorithm (Och, 2003) is currently the most popular way to tune the parameters of a statistical machine translation (MT) system. MERT is well-understood, easy to implement, and runs quickly, but can behave erratically and does not scale beyond a handful of features. This lack of scalability is a significant weakness, as it inhibits systems from using more than a couple dozen features to discriminate between candidate translations and stymies feature development innovation.

Several researchers have attempted to address this weakness. Recently, Watanabe et al. (2007) and Chiang et al. (2008b) have developed tuning methods using the MIRA algorithm (Crammer and Singer, 2003) as a nucleus. The MIRA technique of Chiang et al. has been shown to perform well on large-scale tasks with hundreds or thousands of features (2009). However, the technique is complex and architecturally quite different from MERT. Tellingly, in the entire proceedings of ACL 2010 (Hajič et al., 2010), only one paper describing a statistical MT system cited the use of MIRA for tuning (Chiang, 2010), while 15 used MERT.[1]

Here we propose a simpler approach to tuning that scales similarly to high-dimensional feature spaces. We cast tuning as a ranking problem (Chen et al., 2009), where the explicit goal is to learn to correctly rank candidate translations. Specifically, we follow the *pairwise approach* to ranking (Herbrich et al., 1999; Freund et al., 2003; Burges et al., 2005; Cao et al., 2007), in which the ranking problem is reduced to the binary classification task of deciding between candidate translation pairs.

Of primary concern to us is the ease of adoption of our proposed technique. Because of this, we adhere as closely as possible to the established MERT architecture and use freely available machine learning software. The end result is a technique that scales and performs just as well as MIRA-based tuning, but which can be implemented in a couple of hours by anyone with an existing MERT implementation. Mindful that many would-be enhancements to the

[1] The remainder either did not specify their tuning method (though a number of these used the Moses toolkit (Koehn et al., 2007), which uses MERT for tuning) or, in one case, set weights by hand.

state-of-the-art are false positives that only show improvement in a narrowly defined setting or with limited data, we validate our claims on both syntax and phrase-based systems, using multiple language pairs and large data sets.

We describe tuning in abstract and somewhat formal terms in Section 2, describe the MERT algorithm in the context of those terms and illustrate its scalability issues via a synthetic experiment in Section 3, introduce our pairwise ranking optimization method in Section 4, present numerous large-scale MT experiments to validate our claims in Section 5, discuss some related work in Section 6, and conclude in Section 7.

2 Tuning

In Figure 1, we show an example *candidate space*, defined as a tuple $\langle \Delta, I, J, f, e, \mathbf{x} \rangle$ where:

- Δ is a positive integer referred to as the *dimensionality* of the space
- I is a (possibly infinite) set of positive integers, referred to as *sentence indices*
- J maps each sentence index to a (possibly infinite) set of positive integers, referred to as *candidate indices*
- f maps each sentence index to a sentence from the source language
- e maps each pair $\langle i, j \rangle \in I \times J(i)$ to the j^{th} target-language *candidate translation* of source sentence $f(i)$
- $\mathbf{x}$ maps each pair $\langle i, j \rangle \in I \times J(i)$ to a Δ-dimension *feature vector* representation of $e(i, j)$

The example candidate space has two source sentences, three candidate translations for each source sentence, and feature vectors of dimension 2. It is an example of a *finite* candidate space, defined as a candidate space for which I is finite and J maps each index of I to a finite set.

A *policy* of candidate space $\langle \Delta, I, J, f, e, \mathbf{x} \rangle$ is a function that maps each member $i \in I$ to a member of $J(i)$. A policy corresponds to a choice of one candidate translation for each source sentence. For the example in Figure 1, policy $p_1 = \{1 \mapsto 2, 2 \mapsto 3\}$ corresponds to the choice of "he does not go" for the first source sentence and "I do not go" for the second source sentence. Obviously some policies are better than others. Policy $p_2 = \{1 \mapsto 3, 2 \mapsto 1\}$ corresponds to the inferior translations "she not go" and "I go not."

We assume the MT system distinguishes between policies using a scoring function for candidate translations of the form $h_{\mathbf{w}}(i, j) = \mathbf{w} \cdot \mathbf{x}(i, j)$, where $\mathbf{w}$ is a weight vector of the same dimension as feature vector $\mathbf{x}(i, j)$. This scoring function extends to a policy p by summing the cost of each of the policy's candidate translations: $H_{\mathbf{w}}(p) = \sum_{i \in I} h_{\mathbf{w}}(i, p(i))$. As can be seen in Figure 1, using $\mathbf{w} = [-2, 1]$, $H_{\mathbf{w}}(p_1) = 9$ and $H_{\mathbf{w}}(p_2) = -8$.

The goal of tuning is to learn a weight vector $\mathbf{w}$ such that $H_{\mathbf{w}}(p)$ assigns a high score to good policies, and a low score to bad policies.[2] To do so, we need information about which policies are good and which are bad. This information is provided by a "gold" scoring function G that maps each policy to a real-valued score. Typically this gold function is BLEU (Papineni et al., 2002), though there are several common alternatives (Lavie and Denkowski, 2009; Melamed et al., 2003; Snover et al., 2006; Chiang et al., 2008a).

We want to find a weight vector $\mathbf{w}$ such that $H_{\mathbf{w}}$ behaves "similarly" to G on a candidate space s. We assume a loss function $l_s(H_{\mathbf{w}}, G)$ which returns the real-valued loss of using scoring function $H_{\mathbf{w}}$ when the gold scoring function is G and the candidate space is s. Thus, we may say the goal of tuning is to find the weight vector $\mathbf{w}$ that minimizes loss.

3 MERT

In general, the candidate space may have infinitely many source sentences, as well as infinitely many candidate translations per source sentence. In practice, tuning optimizes over a finite subset of source sentences[3] and a finite subset of candidate translations as well. The classic tuning architecture used in the dominant MERT approach (Och, 2003) forms the translation subset and learns weight vector $\mathbf{w}$ via

[2] Without loss of generality, we assume that a higher score indicates a better translation.

[3] See Section 5.2 for the tune sets used in this paper's experiments.

Source Sentence		Candidate Translations				
i	$f(i)$	j	$e(i,j)$	$\mathbf{x}(i,j)$	$h_{\mathbf{w}}(i,j)$	$g(i,j)$
1	"il ne va pas"	1	"he goes not"	[2 4]	0	0.28
		2	"he does not go"	[3 8]	2	0.42
		3	"she not go"	[6 1]	-11	0.12
2	"je ne vais pas"	1	"I go not"	[-3 -3]	3	0.15
		2	"we do not go"	[1 -5]	-7	0.18
		3	"I do not go"	[-5 -3]	7	0.34

Figure 1: Example candidate space of dimensionality 2. Note: $I = \{1, 2\}$, $J(1) = J(2) = \{1, 2, 3\}$. We also show a local scoring function $h_{\mathbf{w}}(i, j)$ (where $\mathbf{w} = [-2, 1]$) and a local gold scoring function $g(i, j)$.

```
Algorithm TUNE($s$, $G$):
initialize pool: let $s' = \langle \Delta, I', J', f, e, \mathbf{x} \rangle$, where $I' \subseteq I$ and $J' = \emptyset$
for the desired number of iterations do
    candidate generation: choose index pairs $(i, j)$; for each, add $j$ to $J'(i)$
    optimization: find vector $\mathbf{w}$ that minimizes $l_{s'}(H_{\mathbf{w}}, G)$
return $\mathbf{w}$
```

Figure 2: Schema for iterative tuning of base candidate space $s = \langle \Delta, I, J, f, e, \mathbf{x} \rangle$ w.r.t. gold function G.

a feedback loop consisting of two phases. Figure 2 shows the pseudocode. During *candidate generation*, candidate translations are selected from a base candidate space s and added to a finite candidate space s' called the *candidate pool*. During *optimization*, the weight vector $\mathbf{w}$ is optimized to minimize loss $l_{s'}(H_{\mathbf{w}}, G)$.

For its candidate generation phase, MERT generates the k-best candidate translations for each source sentence according to $h_{\mathbf{w}}$, where $\mathbf{w}$ is the weight vector from the previous optimization phase (or an arbitrary weight vector for the first iteration).

For its optimization phase, MERT defines the loss function as follows:

$$l_s(H_{\mathbf{w}}, G) = \max_p G(p) - G(\arg\max_p H_{\mathbf{w}}(p))$$

In other words, it prefers weight vectors $\mathbf{w}$ such that the gold function G scores $H_{\mathbf{w}}$'s best policy as highly as possible (if $H_{\mathbf{w}}$'s best policy is the same as G's best policy, then there is zero loss). Typically the optimization phase is implemented using Och's line optimization algorithm (2003).

MERT has proven itself effective at tuning candidate spaces with low dimensionality. However, it is often claimed that MERT does not scale well with dimensionality. To test this claim, we devised the following synthetic data experiment:

1. We created a gold scoring function G that is *also* a linear function of the same form as $H_{\mathbf{w}}$, i.e., $G(p) = H_{\mathbf{w}^*}(p)$ for some gold weight vector $\mathbf{w}^*$. Under this assumption, the role of the optimization phase reduces to learning back the gold weight vector $\mathbf{w}^*$.

2. We generated a Δ-dimensionality candidate pool with 500 source "sentences" and 100 candidate "translations" per sentence. We created the corresponding feature vectors by drawing Δ random real numbers uniformly from the interval $[0, 500]$.

3. We ran MERT's line optimization on this synthetic candidate pool and compared the learned weight vector $\mathbf{w}$ to the gold weight vector $\mathbf{w}^*$ using cosine similarity.

We used line optimization in the standard way, by generating 20 random starting weight vectors and hill-climbing on each independently until no further progress is made, then choosing the final weight vector that minimizes loss. We tried various dimensionalities from 10 to 1000. We repeated each setting three times, generating different random data each time. The results in Figure 3 indicate that as the dimensionality of the problem increases MERT rapidly loses the ability to learn $\mathbf{w}^*$. Note that this synthetic problem is considerably easier than a real MT scenario, where the data is noisy and interdependent, and the gold scoring function is nonlinear. If

MERT cannot scale in this simple scenario, it has little hope of succeeding in a high-dimensionality deployment scenario.

4 Optimization via Pairwise Ranking

We would like to modify MERT so that it scales well to high-dimensionality candidate spaces. The most prominent example of a tuning method that performs well on high-dimensionality candidate spaces is the MIRA-based approach used by Watanabe et al. (2007) and Chiang et al. (2008b; 2009). Unfortunately, this approach requires a complex architecture that diverges significantly from the MERT approach, and consequently has not been widely adopted. Our goal is to achieve the same performance with minimal modification to MERT.

With MERT as a starting point, we have a choice: modify candidate generation, optimization, or both. Although alternative candidate generation methods have been proposed (Macherey et al., 2008; Chiang et al., 2008b; Chatterjee and Cancedda, 2010), we will restrict ourselves to MERT-style candidate generation, in order to minimize divergence from the established MERT tuning architecture. Instead, we focus on the optimization phase.

4.1 Basic Approach

While intuitive, the MERT optimization module focuses attention on $H_{\mathbf{w}}$'s best policy, and not on its overall prowess at ranking policies. We will create an optimization module that directly addresses $H_{\mathbf{w}}$'s ability to rank policies in the hope that this more holistic approach will generalize better to unseen data.

Assume that the gold scoring function G decomposes in the following way:

$$G(p) = \sum_{i \in I} g(i, p(i)) \tag{1}$$

where $g(i,j)$ is a local scoring function that scores the single candidate translation $e(i,j)$. We show an example g in Figure 1. For an arbitrary pair of candidate translations $e(i,j)$ and $e(i,j')$, the local gold function g tells us which is the better translation. Note that this induces a ranking on the candidate translations for each source sentence.

We follow the *pairwise approach* to ranking (Herbrich et al., 1999; Freund et al., 2003; Burges et al., 2005; Cao et al., 2007). In the pairwise approach, the learning task is framed as the classification of candidate pairs into two categories: correctly ordered and incorrectly ordered. Specifically, for candidate translation pair $e(i,j)$ and $e(i,j')$, we want: $g(i,j) > g(i,j') \Leftrightarrow h_{\mathbf{w}}(i,j) > h_{\mathbf{w}}(i,j')$. We can re-express this condition:

$$\begin{aligned} g(i,j) > g(i,j') &\Leftrightarrow h_{\mathbf{w}}(i,j) > h_{\mathbf{w}}(i,j') \\ &\Leftrightarrow h_{\mathbf{w}}(i,j) - h_{\mathbf{w}}(i,j') > 0 \\ &\Leftrightarrow \mathbf{w} \cdot \mathbf{x}(i,j) - \mathbf{w} \cdot \mathbf{x}(i,j') > 0 \\ &\Leftrightarrow \mathbf{w} \cdot (\mathbf{x}(i,j) - \mathbf{x}(i,j')) > 0 \end{aligned}$$

Thus optimization reduces to a classic binary classification problem. We create a labeled training instance for this problem by computing difference vector $\mathbf{x}(i,j) - \mathbf{x}(i,j')$, and labeling it as a positive or negative instance based on whether, respectively, the first or second vector is superior according to gold function g. To ensure balance, we consider both possible difference vectors from a pair. For example, given the candidate space of Figure 1, since $g(1,1) > g(1,3)$, we would add $([-4,3],+)$ and $([4,-3],-)$ to our training set. We can then feed this training data directly to any off-the-shelf classification tool that returns a linear classifier, in order to obtain a weight vector $\mathbf{w}$ that optimizes the above condition. This weight vector can then be used directly by the MT system in the subsequent candidate generation phase. The exact loss function $l_{s'}(H_{\mathbf{w}}, G)$ optimized depends on the choice of classifier.[4]

Typical approaches to pairwise ranking enumerate all difference vectors as training data. For tuning however, this means $O(|I| * J_{max}^2)$ vectors, where J_{max} is the cardinality of the largest $J(i)$. Since I and J_{max} commonly range in the thousands, a full enumeration would produce billions of feature vectors. Out of tractability considerations, we sample from the space of difference vectors, using the sampler template in Figure 4. For each source sentence i, the sampler generates Γ candidate translation pairs $\langle j, j' \rangle$, and accepts each pair with probability $\alpha_i(|g(i,j) - g(i,j')|)$. Among the accepted pairs, it keeps the Ξ with greatest g differential, and adds their difference vectors to the training data.[5]

[4] See (Chen et al., 2009) for a brief survey.

[5] The intuition for biasing toward high score differential is

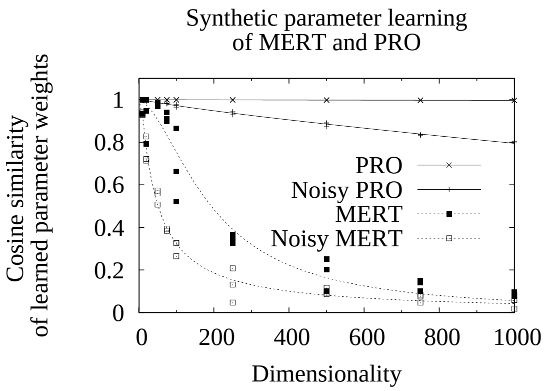

Figure 3: Result of synthetic data learning experiment for MERT and PRO, with and without added noise. As the dimensionality increases MERT is unable to learn the original weights but PRO still performs adequately.

```
**Algorithm** SAMPLER$_{s,g}$( $\Gamma$, $\Xi$, $i$, $\alpha_i$ ):
$V = \langle\rangle$
**for** $\Gamma$ samplings **do**
    Choose $\langle j, j'\rangle \in J(i) \times J(i)$ uniformly at random.
    With probability $\alpha_i(|g(i,j)\text{-}g(i,j')|)$, add $(\mathbf{x}(i,j), \mathbf{x}(i,j'), |g(i,j)\text{-}g(i,j')|)$ to $V$.
Sort $V$ decreasingly by $|g(i,j)\text{-}g(i,j')|$.
**return** $(\mathbf{x}(i,j) - \mathbf{x}(i,j'), \mathsf{sign}(g(i,j)\text{-}g(i,j')))$ and $(\mathbf{x}(i,j')\text{-}\mathbf{x}(i,j), \mathsf{sign}(g(i,j')\text{-}g(i,j)))$ for each of the first $\Xi$ members of $V$.
```

Figure 4: Pseudocode for our sampler. Arguments: $s = \langle\Delta, I, J, f, e, \mathbf{x}\rangle$ is a finite candidate space; g is a scoring function; Γ, Ξ, i are nonnegative integers; α_i is a function from the nonnegative real numbers to the real interval $[0, 1]$.

4.2 Scalability

We repeated the scalability study from Section 3, now using our pairwise ranking optimization (hereafter, PRO) approach. Throughout all experiments with PRO we choose $\Gamma = 5000$, $\Xi = 50$, and the following step function α for each α_i: [6]

$$\alpha(n) = \begin{cases} 0 \text{ if } n < 0.05 \\ 1 \text{ otherwise} \end{cases}$$

We used MegaM (Daumé III, 2004) as a binary classifier in our contrasting synthetic experiment and ran it "out of the box," i.e., with all default settings for binary classification.[7] Figure 3 shows that PRO is able to learn $\mathbf{w}^*$ nearly perfectly at all dimensionalities from 10 to 1000.

As noted previously, though, this is a rather simple task. To encourage a disconnect between g and $h_{\mathbf{w}}$ and make the synthetic scenario look more like MT reality, we repeated the synthetic experiments but added noise to each feature vector, drawn from a zero-mean Gaussian with a standard deviation of 500. The results of the noisy synthetic experiments, also in Figure 3 (the lines labeled "Noisy"), show that the pairwise ranking approach is less successful than before at learning $\mathbf{w}^*$ at high dimensionality, but still greatly outperforms MERT.

4.3 Discussion

The idea of learning from difference vectors also lies at the heart of the MIRA-based approaches (Watanabe et al., 2007; Chiang et al., 2008b) and the approach of Roth et al. (2010), which, similar to our method, uses sampling to select vectors. Here, we isolate these aspects of those approaches to create a simpler tuning technique that closely mirrors the ubiquitous MERT architecture. Among other simplifications, we abstract away the choice of MIRA as the classification method (our approach can use any classification technique that learns a separating hyperplane), and we eliminate the need for oracle translations.

An important observation is that BLEU does not satisfy the decomposability assumption of Equation (1). An advantage of MERT is that it can directly optimize for non-decomposable scoring functions like BLEU. In our experiments, we use the BLEU+1 approximation to BLEU (Lin and Och, 2004) to determine class labels. We will nevertheless use BLEU to evaluate the trained systems.

that our primary goal is to ensure good translations are preferred to bad translations, and not to tease apart small differences.

[6] We obtained these parameters by trial-and-error experimentation on a single MT system (Urdu-English SBMT), then held them fixed throughout our experiments. We obtained similar results using $\Gamma = \Xi = 100$, and for each α_i, a logistic sigmoid function centered at the mean g differential of candidate translation pairs for the i^{th} source sentence. This alternative approach has the advantage of being agnostic about which gold scoring function is used.

[7] With the sampling settings previously described and MegaM as our classifier we were able to optimize two to three times faster than with MERT's line optimization.

PBMT					SBMT				
Language	Experiment		BLEU		Language	Experiment		BLEU	
	feats	method	tune	test		feats	method	tune	test
Urdu-English	base	MERT	20.5	17.7	Urdu-English	base	MERT	23.4	21.4
		MIRA	20.5	17.9			MIRA	23.6	22.3
		PRO	20.4	18.2			PRO	23.4	22.2
	ext	MIRA	21.8	17.8		ext	MIRA	25.2	22.8
		PRO	21.6	18.1			PRO	24.2	22.8
Arabic-English	base	MERT	46.8	41.2	Arabic-English	base	MERT	44.7	39.0
		MIRA	47.0	41.1			MIRA	44.6	39.0
		PRO	46.9	41.1			PRO	44.5	39.0
	ext	MIRA	47.5	41.7		ext	MIRA	45.8	39.8
		PRO	48.5	41.9			PRO	45.9	40.3
Chinese-English	base	MERT	23.8	22.2	Chinese-English	base	MERT	25.5	22.7
		MIRA	24.1	22.5			MIRA	25.4	22.9
		PRO	23.8	22.5			PRO	25.5	22.9
	ext	MIRA	24.8	22.6		ext	MIRA	26.0	23.3
		PRO	24.9	22.7			PRO	25.6	23.5

Table 1: Machine translation performance for the experiments listed in this paper. Scores are case-sensitive IBM BLEU. For every choice of system, language pair, and feature set, PRO performs comparably with the other methods.

5 Experiments

We now turn to real machine translation conditions to validate our thesis: We can cleanly replace MERT's line optimization with pairwise ranking optimization and immediately realize the benefits of high-dimension tuning. We now detail the three language pairs, two feature scenarios, and two MT models used for our experiments. For each language pair and each MT model we used MERT, MIRA, and PRO to tune with a standard set of baseline features, and used the latter two methods to tune with an extended set of features.[8] At the end of every experiment we used the final feature weights to decode a held-out test set and evaluated it with case-sensitive BLEU. The results are in Table 1.

5.1 Systems

We used two systems, each based on a different MT model. Our syntax-based system (hereafter, SBMT) follows the model of Galley et al. (2004). Our phrase-based system (hereafter, PBMT) follows the model of Och and Ney (2004). In both systems we learn alignments with GIZA++ (Och and Ney, 2000) using IBM Model 4; for Urdu-English and Chinese-English we merged alignments with the refined method, and for Arabic-English we merged with the union method.

5.2 Data

Table 2 notes the sizes of the datasets used in our experiments. All tune and test data have four English reference sets for the purposes of scoring.

Data		U-E	A-E	C-E
Train	lines	515K	6.5M	7.9M
	words	2.2M	175M	173M
Tune	lines	923	1994	1615
	words	16K	65K	42K
Test	lines	938	1357	1357
	words	18K	47K	37K

Table 2: Data sizes for the experiments reported in this paper (English words shown).

[8] MERT could not run to a satisfactory completion in any extended feature scenario; as implied in the synthetic data experiment of Section 3, the algorithm makes poor choices for its weights and this leads to low-quality k-best lists and dismal performance, near 0 BLEU in every iteration.

Class	**Urdu-English**				**Arabic-English**				**Chinese-English**			
	PBMT		**SBMT**		**PBMT**		**SBMT**		**PBMT**		**SBMT**	
	base	**ext**	**base**	**ext**	**base**	**ext**	**base**	**ext**	**base**	**ext**	**base**	**ext**
baseline	15	15	19	19	15	15	19	19	15	15	19	19
target word	–	51	–	50	–	51	–	50	–	51	–	299
discount	–	11	–	11	–	11	–	10	–	11	–	10
node count	–	–	–	99	–	–	–	138	–	–	–	96
rule overlap	–	–	–	98	–	–	–	136	–	–	–	93
word pair	–	2110	–	–	–	6193	–	–	–	1688	–	–
phrase length	–	63	–	–	–	63	–	–	–	63	–	–
total	15	2250	19	277	15	6333	18	352	15	1828	19	517

Table 3: Summary of features used in experiments in this paper.

5.2.1 Urdu-English

The training data for Urdu-English is that made available in the constrained track in the NIST 2009 MT evaluation. This includes many lexicon entries and other single-word data, which accounts for the large number of lines relative to word count. The NIST 2008 evaluation set, which contains newswire and web data, is split into two parts; we used roughly half each for tune and test. We trained a 5-gram English language model on the English side of the training data.

5.2.2 Arabic-English

The training data for Arabic English is that made available in the constrained track in the NIST 2008 MT evaluation. The tune set, which contains only newswire data, is a mix from NIST MT evaluation sets from 2003–2006 and from GALE development data. The test set, which contains both web and newswire data, is the evaluation set from the NIST 2008 MT evaluation. We trained a 4-gram English language model on the English side of the training data.

5.2.3 Chinese-English

For Chinese-English we used 173M words of training data from GALE 2008. For SBMT we used a 32M word subset for extracting rules and building a language model, but used the entire training data for alignments, and for all PBMT training. The tune and test sets both contain web and newswire data. The tune set is selected from NIST MT evaluation sets from 2003–2006. The test set is the evaluation set from the NIST 2008 MT evaluation. We trained a 3-gram English language model on the English side of the training data.

5.3 Features

For each of our systems we identify two feature sets: *baseline*, which correspond to the typical small feature set reported in current MT literature, and *extended*, a superset of baseline, which adds hundreds or thousands of features. Specifically, we use 15 baseline features for PBMT, similar to the baseline features described by Watanabe et al. (2007). We use 19 baseline features for SBMT, similar to the baseline features described by Chiang et al. (2008b).

We used the following feature classes in SBMT and PBMT extended scenarios:

- Discount features for rule frequency bins (cf. Chiang et al. (2009), Section 4.1)
- Target word insertion features[9]

We used the following feature classes in SBMT extended scenarios only (cf. Chiang et al. (2009), Section 4.1):[10]

- Rule overlap features
- Node count features

[9] For Chinese-English and Urdu-English SBMT these features only fired when the inserted target word was unaligned to any source word.

[10] The parser used for Arabic-English had a different nonterminal set than that used for the other two SBMT systems, accounting for the wide disparity in feature count for these feature classes.

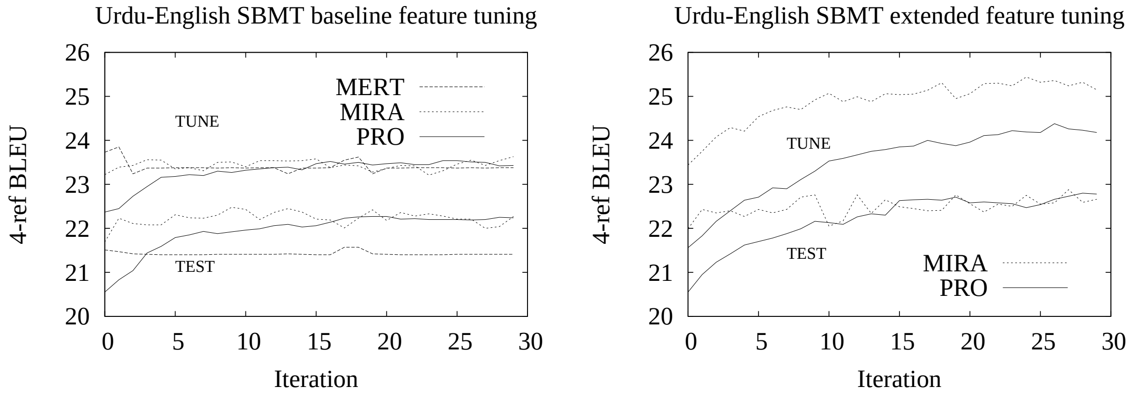

Figure 5: Comparison of MERT, PRO, and MIRA on tuning Urdu-English SBMT systems, and test results at every iteration. PRO performs comparably to MERT and MIRA.

We used the following feature classes in PBMT extended scenarios only:

- Unigram word pair features for the 80 most frequent words in both languages plus tokens for unaligned and all other words (cf. Watanabe et al. (2007), Section 3.2.1)[11]
- Source, target, and joint phrase length features from 1 to 7, e.g. "tgt=4", "src=2", and "src/tgt=2,4"

The feature classes and number of features used within those classes for each language pair are summarized in Table 3.

5.4 Tuning settings

Each of the three approaches we compare in this study has various details associated with it that may prove useful to those wishing to reproduce our results. We list choices made for the various tuning methods here, and note that all our decisions were made in keeping with best practices for each algorithm.

5.4.1 MERT

We used David Chiang's CMERT implementation of MERT that is available with the Moses system (Koehn et al., 2007). We ran MERT for up to 30 iterations, using $k = 1500$, and stopping early when the accumulated k-best list does not change in an iteration. In every tuning iteration we ran MERT once with weights initialized to the last iteration's chosen weight set and 19 times with random weights, and chose the the best of the 20 ending points according to G on the development set. The G we optimize is tokenized, lower-cased 4-gram BLEU (Papineni et al., 2002).

5.4.2 MIRA

We for the most part follow the MIRA algorithm for machine translation as described by Chiang et al. (2009)[12] but instead of using the 10-best of each of the best $h_{\mathbf{w}}$, $h_{\mathbf{w}}$ $+g$, and $h_{\mathbf{w}}$-g, we use the 30-best according to $h_{\mathbf{w}}$.[13] We use the same sentence-level BLEU calculated in the context of previous 1-best translations as Chiang et al. (2008b; 2009). We ran MIRA for 30 iterations.

5.4.3 PRO

We used the MegaM classifier and sampled as described in Section 4.2. As previously noted, we used BLEU+1 (Lin and Och, 2004) for g. MegaM was easy to set up and ran fairly quickly, however any linear binary classifier that operates on real-valued features can be used, and in fact we obtained simi-

[11] This constitutes 6,723 features in principle ($82^2 - 1$ since "unaligned-unaligned" is not considered) but in practice far fewer co-occurrences were seen. Table 3 shows the number of actual unigram word pair features observed in data.

[12] and acknowledge the use of David Chiang's code

[13] This is a more realistic scenario for would-be implementers of MIRA, as obtaining the so-called "hope" and "fear" translations from the lattice or forest is significantly more complicated than simply obtaining a k-best list. Other tests comparing these methods have shown between 0.1 to 0.3 BLEU drop using 30-best $h_{\mathbf{w}}$ on Chinese-English (Wang, 2011).

lar results using the support vector machine module of WEKA (Hall et al., 2009) as well as the Stanford classifier (Manning and Klein, 2003). We ran for up to 30 iterations and used the same k and stopping criterion as was used for MERT, though variability of sampling precluded list convergence.

While MERT and MIRA use each iteration's final weights as a starting point for hill-climbing the next iteration, the pairwise ranking approach has no explicit tie to previous iterations. To incorporate such stability into our process we interpolated the weights $\mathbf{w}'$ learned by the classifier in iteration t with those from iteration $t-1$ by a factor of Ψ, such that $\mathbf{w}_t = \Psi \cdot \mathbf{w}' + (1-\Psi) \cdot \mathbf{w}_{t-1}$. We found $\Psi = 0.1$ gave good performance across the board.

5.5 Discussion

We implore the reader to avoid the natural tendency to compare results using baseline vs. extended features or between PBMT and SBMT on the same language pair. Such discussions are indeed interesting, and could lead to improvements in feature engineering or sartorial choices due to the outcome of wagers (Goodale, 2008), but they distract from our thesis. As can be seen in Table 1, for each of the 12 choices of system, language pair, and feature set, the PRO method performed nearly the same as or better than MIRA and MERT on test data.

In Figure 5 we show the tune and test BLEU using the weights learned at every iteration for each Urdu-English SBMT experiment. Typical of the rest of the experiments, we can clearly see that PRO appears to proceed more monotonically than the other methods. We quantified PRO's stability as compared to MERT by repeating the Urdu-English baseline PBMT experiment five times with each configuration. The tune and test BLEU at each iteration is depicted in Figure 6. The standard deviation of the final test BLEU of MERT was 0.13 across the five experiment instances, while PRO had a standard deviation of just 0.05.

6 Related Work

Several works (Shen et al., 2004; Cowan et al., 2006; Watanabe et al., 2006) have used discriminative techniques to re-rank k-best lists for MT. Tillmann and Zhang (2005) used a customized form of multi-class stochastic gradient descent to learn feature weights for an MT model. Och and Ney (2002) used maximum entropy to tune feature weights but did not compare pairs of derivations. Ittycheriah and Roukos (2005) used a maximum entropy classifier to train an alignment model using hand-labeled data. Xiong et al. (2006) also used a maximum entropy classifier, in this case to train the reordering component of their MT model. Lattice- and hypergraph-based variants of MERT (Macherey et al., 2008; Kumar et al., 2009) are more stable than traditional MERT, but also require significant engineering efforts.

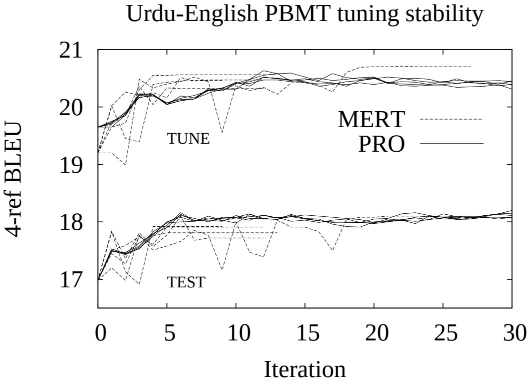

Figure 6: Tune and test curves of five repetitions of the same Urdu-English PBMT baseline feature experiment. PRO is more stable than MERT.

7 Conclusion

We have described a simple technique for tuning an MT system that is on par with the leading techniques, exhibits reliable behavior, scales gracefully to high-dimension feature spaces, and is remarkably easy to implement. We have demonstrated, via a litany of experiments, that our claims are valid and that this technique is widely applicable. It is our hope that the adoption of PRO tuning leads to fewer headaches during tuning and motivates advanced MT feature engineering research.

Acknowledgments

Thanks to Markus Dreyer, Kevin Knight, Saiyam Kohli, Greg Langmead, Daniel Marcu, Dragos Munteanu, and Wei Wang for their assistance. Thanks also to the anonymous reviewers, especially the reviewer who implemented PRO during the review period and replicated our results.

References

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 89–96, Bonn, Germany. ACM.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136, Corvalis, OR.

Samidh Chatterjee and Nicola Cancedda. 2010. Minimum error rate training by sampling the translation lattice. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Cambridge, MA, October. Association for Computational Linguistics.

Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. 2009. Ranking measures and loss functions in learning to rank. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 315–323.

David Chiang, Steve DeNeefe, Yee Seng Chan, and Hwee Tou Ng. 2008a. Decomposability of translation metrics for improved evaluation and efficient algorithms. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 610–619, Honolulu, HI, October. Association for Computational Linguistics.

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

David Chiang, Kevin Knight, and Wei Wang. 2009. 11,001 new features for statistical machine translation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 218–226, Boulder, CO, June. Association for Computational Linguistics.

David Chiang. 2010. Learning to translate with source and target syntax. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1443–1452, Uppsala, Sweden, July. Association for Computational Linguistics.

Brooke Cowan, Ivona Kučerová, and Michael Collins. 2006. A discriminative model for tree-to-tree translation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 232–241, Sydney, Australia, July. Association for Computational Linguistics.

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

Hal Daumé III. 2004. Notes on CG and LM-BFGS optimization of logistic regression. Paper available at `http://pub.hal3.name#daume04cg-bfgs`, implementation available at `http://hal3.name/megam/`, August.

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

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *HLT-NAACL 2004: Main Proceedings*, pages 273–280, Boston, MA, May. Association for Computational Linguistics.

Gloria Goodale. 2008. Language Weaver: fast in translation. *The Christian Science Monitor*, October 1. `http://www.csmonitor.com/Innovation/Tech-Culture/2008/1001/language-weaver-fast-in-translation`.

Jan Hajič, Sandra Carberry, Stephen Clark, and Joakim Nivre, editors. 2010. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Uppsala, Sweden, July.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1).

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. 1999. Support vector learning for ordinal regression. In *Proceedings of the 1999 International Conference on Artificial Neural Networks*, pages 97–102.

Abraham Ittycheriah and Salim Roukos. 2005. A maximum entropy word aligner for Arabic-English machine translation. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 89–96, Vancouver, Canada, October. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–

180, Prague, Czech Republic, June. Association for Computational Linguistics.

Shankar Kumar, Wolfgang Macherey, Chris Dyer, and Franz Och. 2009. Efficient minimum error rate training and minimum bayes-risk decoding for translation hypergraphs and lattices. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 163–171, Suntec, Singapore, August. Association for Computational Linguistics.

Alon Lavie and Michael J. Denkowski. 2009. The METEOR metric for automatic evaluation of machine translation. *Machine Translation*, 23(2–3):105–115, September.

Chin-Yew Lin and Franz Josef Och. 2004. ORANGE: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, pages 501–507, Geneva, Switzerland, August.

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

Christopher Manning and Dan Klein. 2003. Optimization, maxent models, and conditional estimation without magic. Tutorial at HLT-NAACL 2003 and ACL 2003.

I. Dan Melamed, Ryan Green, and Joseph P. Turian. 2003. Precision and recall of machine translation. In *Companion Volume of the Proceedings of HLT-NAACL 2003 - Short Papers*, pages 61–63, Edmonton, Canada, May–June. Association for Computational Linguistics.

Franz Och and Hermann Ney. 2000. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 440–447, Hong Kong, October.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 295–302, Philadelphia, PA, July. Association for Computational Linguistics.

Franz Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

Franz Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Sapporo, Japan, July. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, PA, July. Association for Computational Linguistics.

Benjamin Roth, Andrew McCallum, Marc Dymetman, and Nicola Cancedda. 2010. Machine translation using overlapping alignments and samplerank. In *Proceedings of Association for Machine Translation in the Americas*, Denver, CO.

Libin Shen, Anoop Sarkar, and Franz Josef Och. 2004. Discriminative reranking for machine translation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 177–184, Boston, MA, May 2 - May 7. Association for Computational Linguistics.

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

Christoph Tillmann and Tong Zhang. 2005. A localized prediction model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 557–564, Ann Arbor, MI, June. Association for Computational Linguistics.

Wei Wang. 2011. Personal communication.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2006. NTT statistical machine translation for IWSLT 2006. In *Proceedings of IWSLT 2006*, pages 95–102.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 764–773, Prague, Czech Republic, June. Association for Computational Linguistics.

Deyi Xiong, Qun Liu, and Shouxun Lin. 2006. Maximum entropy based phrase reordering model for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 521–528, Sydney, Australia, July. Association for Computational Linguistics.