APRO: All-Pairs Ranking Optimization for MT Tuning

Markus Dreyer*
SDL Research
6060 Center Drive Suite 150
Los Angeles, CA 90045
markus.dreyer@gmail.com

Yuanzhe Dong
SDL Research
6060 Center Drive Suite 150
Los Angeles, CA 90045
ydong@sdl.com

Abstract

We present APRO, a new method for machine translation tuning that can handle large feature sets. As opposed to other popular methods (e.g., MERT, MIRA, PRO), which involve randomness and require multiple runs to obtain a reliable result, APRO gives the same result on any run, given initial feature weights. APRO follows the pairwise ranking approach of PRO (Hopkins and May, 2011), but instead of ranking a small sampled subset of pairs from the k-best list, APRO efficiently ranks *all* pairs. By obviating the need for manually determined sampling settings, we obtain more reliable results. APRO converges more quickly than PRO and gives similar or better translation results.

1 Introduction

Machine translation tuning seeks to find feature weights that maximize translation quality. Recent efforts have focused on methods that scale to large numbers of features (Cherry and Foster, 2012), and among these, PRO has gained popularity (Pairwise Ranking Optimization, Hopkins and May (2011)).

PRO's goal is to find feature weights such that the resulting k-best list entries are ranked in the same way that an evaluation function (e.g., BLEU, Papineni et al. (2002)) ranks them. To do this, it labels pairs of translations for each sentence as *positive* or *negative*, depending on the gold ranking of the two pair elements given by BLEU. A binary classifier is trained on these labeled examples, resulting in new feature weights, and the procedure is iterated. This procedure would ordinarily be too expensive since there are $\mathcal{O}(k^2)$ pairs per sentence, where both k and the number of sentences can be in the thousands, so billions of training examples would be produced per iteration. Therefore, Hopkins and May (2011) use subsampling to consider a small percentage of all pairs per sentence.

We present APRO (All-Pairs Ranking Optimization), a tuning approach that, like PRO, uses pairwise ranking for tuning. Unlike PRO, it is not limited to optimizing a small percentage of pairs per sentence. Based on an efficient ranking SVM formulation (Airola et al. (2011), Lee and Lin (2014)), we find, in each iteration, feature weights that minimize ranking errors for *all* pairs of translations per sentence. This tuning method inherits all the advantages of PRO—it is scalable, effective, easy to implement—and removes its limitations. It does not require meta-tuning of sampling parameters since no sampling is used; it does not need to be run multiple times to obtain reliable results, like MERT (Och, 2003), PRO, MIRA (Chiang et al., 2008) and others, since it uses global optimization and is deterministic given initial feature weights; and it converges quickly.

2 Notation and Definitions

For both PRO and APRO, we use the following definitions: A tuning dataset contains S source sentences $x^1, \ldots, x^S$. Let $\mathcal{Y}^s$ be the space of all translations of x^s. It contains one or more known reference translations $\boldsymbol{y}^s_+$. Each translation $y^s_i \in \mathcal{Y}^s$ has a fea-

*Markus Dreyer is now at Amazon, Inc., Seattle, WA.

Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pages 1018–1023,
Denver, Colorado, May 31 – June 5, 2015.

ture representation[1] $\boldsymbol{f}(x^s, y_i^s)$, or for short, $\boldsymbol{f}_i^s$, and a linear classification score $h_i^s = \mathbf{w}^T \boldsymbol{f}_i^s$, where $\mathbf{w}$ is a feature weight vector. Given a source sentence x^s, a translation decoder can search (often a subset of) $\mathcal{Y}^s$ and return the k translations $y_1^s, \ldots, y_k^s$ with the highest classification scores. A k-best list is the list of $y_1^s, \ldots, y_k^s, \forall s$. For each translation y_i^s we can obtain an evaluation score $b(y_i^s, \boldsymbol{y}_+^s)$, or for short, b_i^s, which can be the BLEU+1 score (Lin and Och, 2004).[2] For a given source sentence x^s, let (i, j) denote a pair of translations (y_i^s, y_j^s).

3 PRO

We now describe PRO, before constrasting it with our new approach, APRO. For each iteration t from $t = 1 \ldots T$, PRO performs the following steps:

1. Given current feature weights $\mathbf{w}_t$, obtain a k-best list, as defined above, from the translation decoder. For each x^s, add to its k-best entries the k-best entries from previous iterations, so that x^s now has k_s translations; the overall list is called an *accumulated* k-best list.

2. For each source sentence x^s, first sample up to Γ candidate pairs from its translations in the k-best list. Less similar pairs are more likely to become candidate pairs. Similarity in a pair (i, j) here means a small absolute difference d_{ij}^s between b_i^s and b_j^s. The most similar pairs ($d_{ij}^s < \beta$) are discarded. Then select the Ξ least similar pairs among the remaining candidate pairs.

3. For each pair (i, j) from the Ξ selected pairs, add the difference vector $(\boldsymbol{f}_i^s - \boldsymbol{f}_j^s)$ with class label 1 if $b_i^s > b_j^s$, otherwise add it with class label -1. Also add $(\boldsymbol{f}_j^s - \boldsymbol{f}_i^s)$ with the opposite label.

4. Train any classifier on the labeled data, resulting in a new weights vector $\mathbf{w}'$. Set $\mathbf{w}_{t+1} = \Psi \cdot \mathbf{w}' + (1 - \Psi) \cdot \mathbf{w}_t$.

Dependencies between tuning iterations are introduced by the use of accumulated k-best lists and the interpolation of weight vectors in step 4, using an interpolation factor Ψ. Translation quality varies with different choices for Γ, Ξ, β, Ψ, see Figure 1. The quality varies even when PRO is run multiple times with the *same* parameters, due to the sampling step. Practitioners would have to perform an expensive grid search multiple times to be sure to obtain good results. APRO seeks to remedy this problem. One could try to improve PRO by experimenting with other pair selection heuristics; APRO circumvents the problem by efficiently selecting *all* pairs.

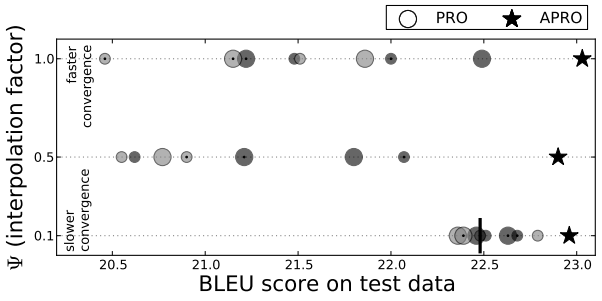

Figure 1: PRO versus APRO (eng-swe) for 3 settings of Ψ. PRO: 8 sampling settings per Ψ setting.[4] APRO: no sampling. Vertical line indicates settings from H&M (2011). Not shown: PRO outlier with BLEU =7.9 at $\Psi = 0.5$.

4 APRO

Our method APRO is, like PRO, a ranking method. We believe that learning to rank is a suitable method for MT tuning because it matches the test-time requirements of correctly predicting the best translations or correctly ranked k-best lists of translations.

Compared to PRO, we simplify the procedure by removing sampling and labeling steps 2 and 3, thereby removing some of PRO's implementation complexity and manually set parameters. We run only two steps, corresponding to PRO's steps 1 and 4: In each tuning iteration, we obtain an accumulated k-best list, then directly find a new $\mathbf{w}'$ that minimizes the loss on that k-best list, which corresponds to PRO's running of a classifier. APRO's classification model is an efficient ranking SVM (Airola et al. (2011), Lee and Lin (2014)), described as follows.

4.1 Model

For each sentence x^s, we define the set of preference pairs as the set of ordered translation pairs for which the evaluation score prefers the first element:

$$\mathcal{P}^s = \{(i, j) \ : \ b_i^s > b_j^s\} \tag{1}$$

[1] For simplicity, we leave out nuisance variables like alignments, segmentations, or parse trees, from this description, which may be part of the feature space.

[2] But see Nakov et al. (2013) for variants.

[4] PRO settings: $\Gamma = \{5k, 8k\} = \{\text{small, large}\}$, $\Xi = \{50, 100\} = \{\text{light, dark}\}$, $\beta = \{.03, .05\} = \{\text{no dot, dot}\}$.

Following Lee and Lin (2014), we define the loss (or, error) of any sentence s as the sum of its pairwise squared hinge losses:

$$L^s_{\mathbf{w}} = \sum_{(i,j)\in\mathcal{P}^s} \max(0, 1-h^s_i+h^s_j)^2 \quad (2)$$

That is, no loss is contributed by preference pairs for which the classification score correctly prefers the first element by a large-enough margin, i.e., $h^s_i \geq h^s_j+1$; all other preference pairs contribute some loss. We seek to find a weight vector that minimizes the regularized overall loss:

$$\mathbf{w}' = \operatorname*{argmin}_{\mathbf{w}} R_{\mathbf{w}} + C\cdot\frac{1}{N}\sum_s L^s_{\mathbf{w}} \quad (3)$$

where $R_{\mathbf{w}} = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is a Gaussian regularizer to prevent overfitting and C a constant controlling the relative regularization amount. We divide by $N = \sum_s k_s$ to account for the increasing sizes of accumulated k-best lists between tuning iterations, which leads to increased sentence losses. If this were not done, the relative amount of regularization would decrease in subsequent iterations of tuning.

Any gradient-based optimization method can be used to find $\mathbf{w}'$. Since the loss is convex, the weights we find given a particular k-best list are optimal. This is different from PRO and Bazrafshan et al. (2012), where the resulting weights depend on the pairs sampled; MIRA, where they depend on the order of sentences processed; and MERT, where optimization is greedy and depends on initial weights.

4.2 Efficient Computation

How do we efficiently compute $L^s_{\mathbf{w}}$ per sentence? In this and the following subsection, we leave out all sentence indices s for ease of notation; it is understood that we operate on a given sentence.

A straightforward algorithm to compute $L_{\mathbf{w}}$ would iterate over all preference pairs $(i,j) \in \mathcal{P}$ and add up their losses (Joachims, 2002). However, since there are $\mathcal{O}(k^2)$ pairs per sentence, with potentially thousands of sentences, this would be extremely inefficient. PRO's solution to this problem is subsampling. The alternative solution we apply is to make the sums over translation pairs efficient by carefully rearranging the terms of the sentence loss, making use of quantities that can be precomputed efficiently (Airola et al. (2011), Lee and Lin (2014)).

Definitions. Let us define those quantities. For a given sentence s, let the set $\mathcal{Q}$ contain all members of $\mathcal{P}$ that contribute a positive loss to the overall loss term:

$$\mathcal{Q} = \{(i,j) : (i,j) \in \mathcal{P} \wedge (1-h_i+h_j > 0)\} \quad (4)$$

We also define an index notation into $\mathcal{Q}$:

$$\mathcal{Q}_{i\bullet} = \{(i,j) \in \mathcal{Q}, \forall j\} \quad q_{i\bullet} = |\mathcal{Q}_{i\bullet}| \quad (5)$$
$$\mathcal{Q}_{\bullet j} = \{(i,j) \in \mathcal{Q}, \forall i\} \quad q_{\bullet j} = |\mathcal{Q}_{\bullet j}| \quad (6)$$
$$r_{i\bullet} = \sum_{(i,j)\in\mathcal{Q}_{i\bullet}} h_j \quad (7)$$

The bullet ($\bullet$) can be read as *any*. Example: $\mathcal{Q}_{\bullet 3}$ contains pairs $\in \mathcal{Q}$ whose second element is translation 3. $q_{i\bullet}$ and $q_{\bullet j}$ denote corresponding set sizes.

Rearrangement. We use these definitions to express the loss as a sum over only $\mathcal{O}(k)$ elements.

First, we simplify the loss expression by summing only over elements from $\mathcal{Q}$, i.e., pairs from $\mathcal{P}$ that contribute a positive loss, so the max becomes unnecessary:

$$L_{\mathbf{w}} = \sum_{(i,j)\in\mathcal{P}} \max(0, 1-h_i+h_j)^2 \quad (8)$$
$$= \sum_{(i,j)\in\mathcal{Q}} (1-h_i+h_j)^2 \quad (9)$$
$$= \sum_{(i,j)\in\mathcal{Q}} h_i^2-2h_i+1+h_j^2+2h_j-2h_ih_j \quad (10)$$

We then use the precomputed quantities defined above to rewrite the sum over $\mathcal{O}(k^2)$ pairs to a sum over just $\mathcal{O}(k)$ elements:

$$L_{\mathbf{w}} = \sum_{i=1}^{k} q_{i\bullet}(h_i^2-2h_i+1)+q_{\bullet i}(h_i^2+2h_i) - 2\, r_{i\bullet}\, h_i \quad (11)$$

This step is described in detail below. Our new formulation is simpler but equivalent to Lee and Lin

(2014). Using order statistics trees (Cormen et al., 2001), the quantities $q_{i\bullet}$, $q_{\bullet i}$, and $r_{i\bullet}$ can be precomputed in $\mathcal{O}(k \log k)$ time (see details in Lee and Lin (2014)). This precomputation, together with the rearranged loss, allows APRO to make efficient weight updates without having to subsample.

Detailed derivation. We explain how to derive Equation 11 from Equation 10.
First, let us define the following equalities:

$$\sum_{(1,j)\in\mathcal{Q}} h_1 = q_{1\bullet}\cdot h_1$$
$$\sum_{(2,j)\in\mathcal{Q}} h_2 = q_{2\bullet}\cdot h_2$$
$$\ldots$$

If we do not fix the first pair element to a particular value, we have:

$$\sum_{(i,j)\in\mathcal{Q}} h_i = \sum_i q_{i\bullet}\cdot h_i \tag{12}$$

Similarly:

$$\sum_{(j,1)\in\mathcal{Q}} h_1 = q_{\bullet 1}\cdot h_1$$
$$\sum_{(j,2)\in\mathcal{Q}} h_2 = q_{\bullet 2}\cdot h_2$$
$$\ldots$$

If we do not fix the second element of each pair to a particular value, we have:

$$\sum_{(j,i)\in\mathcal{Q}} h_i = \sum_i q_{\bullet i}\cdot h_i \tag{13}$$

We split Equation 10 into separate sums and perform a change of variables in the second sum:

$$\begin{aligned} L_{\mathbf{w}} = &\sum_{(i,j)\in\mathcal{Q}} h_i^2 - 2h_i + 1 + \sum_{(j,i)\in\mathcal{Q}} h_i^2 + 2h_i \\ &-2\sum_{(i,j)\in\mathcal{Q}} h_i h_j \end{aligned} \tag{14}$$

We introduce one more equality, where (16) follows from the definition of $r_{i\bullet}$ in Equation 7:

$$\sum_{(i,j)\in\mathcal{Q}} h_i h_j = \sum_i h_i \left(\sum_{(i,j)\in\mathcal{Q}_{i\bullet}} h_j \right) \tag{15}$$

Lang.	Train	Dev	Test
Ara-Eng	14.4M	66K	37K
Chi-Eng	142.9M	61K	29K
Eng-Swe	100.1M	21K	22K
Eng-Fra	100.0M	63K	20K
Ita-Eng	102.8M	21K	20K
Pol-Eng	90.5M	21K	19K

Table 1: Number of words in the used data sets.

$$= \sum_i h_i \; r_{i\bullet} \tag{16}$$

We now use equalities 12, 13, and 16 to arrive at Equation 11:

$$\begin{aligned} L_{\mathbf{w}} = &\sum_i q_{i\bullet}(h_i^2 - 2h_i + 1) + \sum_i q_{\bullet i}(h_i^2 + 2h_i) \\ &-2\sum_i r_{i\bullet}\, h_i \\ = &\sum_i q_{i\bullet}(h_i^2 - 2h_i + 1) + q_{\bullet i}(h_i^2 + 2h_i) \\ &-2 r_{i\bullet}\, h_i \end{aligned}$$

5 Experiments

5.1 Experimental Setup

We validate APRO on 6 diverse language pairs. For each one, we perform HMM-based word alignment (Vogel et al., 1996) and phrase rule extraction on the training data. We use 20 standard features, incl. 8 reordering features, plus the sparse features listed for PBTM systems in Hopkins and May (2011).[5]

For Ara-Eng and Chi-Eng, we use BOLT Y2 data sets.[6] For all other languages, we sample train, dev, and test sets from in-house data. Table 1 describes the different data set sizes. We use 5-gram LMs trained on the target side of the training data; for Ara-Eng and Chi-Eng, we add 2 LMs trained on English Gigaword and other sources.

We tune on dev data. In each tuning run, we use $k = 500$, except for Ara-Eng ($k = 1500$). We use the same weight initialization for every tuning run, where most features are initialized to 0 and some dense features are initialized to 1 or -1. During tuning, we use case-insensitive BLEU+1. We tune for

[5] We use the 500 most frequent words for word pair features.
[6] For ara-eng, a subset of the training data was chosen whose source side has maximum similarity to the test source side.

	PRO		APRO	
	BLEU	LR	BLEU	LR
Ara-Eng	(29.3) 30.7	(0.93) 0.97	(30.3) 30.8	(0.98) 0.99
Chi-Eng	(15.4) 20.8	(0.78) 0.98	(19.2) 20.8	(1.01) 0.98
Eng-Fra	(30.9) 33.0	(0.95) 0.97	(32.7) 33.3	(1.00) 0.99
Eng-Swe	(22.2) 22.4	(1.00) 1.01	(23.1) 23.0	(1.00) 1.00
Ita-Eng	(25.6) 25.3	(1.00) 1.00	(25.2) 25.6	(1.00) 1.00
Pol-Eng	(22.4) 23.0	(0.95) 0.99	(23.3) 23.3	(1.00) 0.99

Table 2: PRO versus APRO after 10 iterations (small in parentheses) and at convergence ($\leq$ 30 iterations). Good results after 10 iterations indicate fast convergence. PRO: mean over 2 runs (average BLEU standard deviation was 0.1); APRO: single run. LR: length ratio.

up to 30 iterations,[7] where we reset the accumulated k-best list after 10 iterations.[8] For PRO, we use Γ=5000, Ξ=50, β=0.05, Ψ=0.1, and (MegaM) regularization strength λ=1 as described in Hopkins and May (2011). For APRO, we use regularization strength C=0.01 and Ψ=1, which effectively removes the weight interpolation step. We repeat each PRO tuning twice and report the mean of length ratios and case-sensitive BLEU scores on test data. For APRO, no repeated runs are necessary; it gives the same result on any run given initial feature weights.

For APRO, we optimize using the implementation by Lee and Lin, which uses a truncated Newton method.[9]

5.2 Results

We measure the runtime of PRO and APRO. For an accumulated k-best list containing s=2,748 sentences with an average k_s=3,600 translation, PRO and APRO take 13 and 8 minutes, respectively. Table 2 shows translation quality after 10 iterations and at convergence. We observe that APRO converges quickly: After running for 10 iterations, it gives higher BLEU scores and better length ratios than PRO for five out of six language pairs. At convergence, PRO has caught up, but for all language pairs APRO performs similar or better.

One of APRO's advantages are stable results: Figure 1 compares PRO and APRO for 3 values of Ψ: For each value, we run PRO eight times with different sampling settings and APRO once. We observe that the different PRO settings result in different BLEU scores. Cherry and Foster (2012) report that they could not find one PRO setting that worked across all language pairs. This suggests that practitioners may have to run expensive grid searches to find optimal PRO performance; this is not necessary with APRO. While PRO performs best with $\Psi = 0.1$, APRO gets good results for Ψ=1, which is the reason for its fast convergence (Table 2).

6 Conclusions

We have presented APRO, a new tuning method for machine translation. Like PRO, APRO is a batch pairwise ranking method, and as such, it inherits PRO's advantages of being effective, scalable to large feature sets and easy to fit into the standard batch MT tuning framework. We remove PRO's sampling step and learn a pairwise ranking over the whole k-best list in $\mathcal{O}(k \log k)$ time. We have shown that PRO's different sampling settings result in different final results; by removing these settings we get more reliable results. We find that PRO's weight interpolation is not necessary for APRO, resulting in faster convergence. At convergence, APRO's translation quality was found to be similar or better than PRO's. APRO's use of global optimization and the lack of randomness lead to more stable tuning with deterministic results.

Acknowledgments

We thank Jonathan May, Mark Hopkins, Bill Byrne, Adria Gispert, Gonzalo Iglesias, Steve DeNeefe and the anonymous reviewers for their valuable comments and suggestions.

References

Antti Airola, Tapio Pahikkala, and Tapio Salakoski. 2011. Training linear ranking SVMs in linearithmic time using red-black trees. *Pattern Recognition Letters*, 32(9):1328–1336.

Marzieh Bazrafshan, Tagyoung Chung, and Daniel Gildea. 2012. Tuning as linear regression. In *Pro-*

[7] Like Hopkins and May (2011), we stop earlier when the accumulated k-best list does not change anymore.

[8] This removes bad translations from early iterations and provides good initial weights for the last 20 iterations. This did not decrease but sometimes increase final performance.

[9] See `http://goo.gl/CVmnoZ`. No change to the software is necessary; but in each iteration it must be called with $C' = \frac{C}{N}$, see Equation 3. We have also experimented with a change to the software that scales the loss of each sentence by the number of translation pairs for that sentence; this did not give reliable BLEU improvements over Equation 3.

ceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 543–547. Association for Computational Linguistics.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 427–436. Association for Computational Linguistics.

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

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. 2001. *Introduction to algorithms*. MIT press Cambridge, 2nd edition.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362. Association for Computational Linguistics.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM.

Ching-Pei Lee and Chih-Jen Lin. 2014. Large-scale linear RankSVM. *Neural computation*, 26(4):781–817.

Chin-Yew Lin and Franz Josef Och. 2004. ORANGE: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of Coling 2004*, pages 501–507, Geneva, Switzerland, Aug 23–Aug 27. COLING.

Preslav Nakov, Francisco Guzmán, and Stephan Vogel. 2013. A tale about PRO and monsters. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 12–17, Sofia, Bulgaria, August. Association for Computational Linguistics.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics.

K. Papineni, S. Roukos, T. Ward, and W. J Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics.