Drem: The AFRL Submission to the WMT15 Tuning Task

Grant Erdmann
Air Force Research Laboratory
grant.erdmann@us.af.mil

Jeremy Gwinnup
SRA International[†]
jeremy.gwinnup.ctr@us.af.mil

Abstract

We define a new algorithm, named "Drem", for tuning the weighted linear model in a statistical machine translation system. Drem has two major innovations. First, it uses scaled derivative-free trust-region optimization rather than other methods' line search or (sub)gradient approximations. Second, it interpolates the decoder output, using information about which decodes produced which translations.

1 Introduction

While searching for the best translation of a text, statistical machine translation systems generate several different quantitative descriptors of the translation, called "features". These features are combined into a single score, by weighting and summing them. A tuning algorithm chooses the weights used in this combination.

MERT (Och, 2003) is the standard tuning algorithm. Many different varieties of error rate training exist, with various techniques, including expectation, line-search, Nelder–Mead simplex (Zhao and Chen, 2009), particle swarm optimization (Suzuki et al., 2011), and stabilization (Foster and Kuhn, 2009). It has been experienced that MERT fails to perform well in larger feature spaces, but recently there has been evidence of a regularized MERT succeeding in high dimensions (Galley et al., 2013).

Other methods have been designed as wholesale replacements for MERT, including MIRA (Chiang et al., 2008), k-best MIRA (Cherry and Foster, 2012), PRO (Hopkins and May, 2011), and Rampion (Gimpel and Smith, 2012).

[†]This work is sponsored by the Air Force Research Laboratory under Air Force contract FA-8650-09-D-6939-029.

MERT's continued use indicates an improved dense-feature optimizer for weighted linear models would be welcome. It is in this context that we introduce our tuning algorithm, named "Drem". In contrast to known varieties of MERT, Drem is not a line-search, simplex, or particle swarm optimization method. It is a derivative-free trust-region method, with several advancements to cater to the particular nature of MT system optimization.

2 Background and definitions

A feature vector $\boldsymbol{f} \in \mathbb{R}^k$ has the relative importance of its components determined by a weight vector $\boldsymbol{w} \in \mathbb{R}^k$. For a weighted linear model, the score used by the decoder to choose the best translation is the scalar product,

$$s(\boldsymbol{w}, \boldsymbol{f}) = \boldsymbol{w}^T \boldsymbol{f} \tag{1}$$

which we call the decoder score. The output of the decoder run on a corpus $\mathcal{C}$ at a weight $\boldsymbol{w}$ is an n-best list $\mathcal{N}(\boldsymbol{w}, \mathcal{C})$. The n-best list can be thought of as a collection of elements of the form $(j, t, \boldsymbol{f})$, where $j \in \mathcal{J}$ is the segment (typically sentence) index within $\mathcal{C}$, t is the text of the translation, and $\boldsymbol{f}$ is the feature vector. The objective of tuning is to choose the weights $\boldsymbol{w}$ such that the translation with the highest decoder score (i.e., the "1-best") will be the segment's best translation. For the test error a human performs an evaluation.

In order to produce the best results on the test set, it is important to optimize some measure of error on a given bilingual development set, $\mathcal{C}_{\text{dev}}$. In tuning we will use the common practice of iteratively decoding and optimizing, and we define $\boldsymbol{w}^{(m)}$ to be the weight used in the m-th decode. During optimization, the development error metric at a weight $\boldsymbol{w}$ (where no decode has been performed) is approximated using only results from prior decodings. At this un-decoded weight we must perform a "pseudo-decoding" to approximate the result of decoding at it.

Proceedings of the Tenth Workshop on Statistical Machine Translation, pages 422–427,
Lisboa, Portugal, 17-18 September 2015.

In Drem we define a pseudo-decoder scoring function s_{dev}, changed from the standard decoder score (1) to incorporate a "fear" of including a translation that was produced by decoding a weight far from the weight under consideration. Several different methods, including MIRA (Chiang et al., 2008), k-best MIRA (Cherry and Foster, 2012), Rampion (Gimpel and Smith, 2012), and Ultraconservative Updating (Liu et al., 2012), and stabilization methods of Foster and Kuhn (2009), include this fear by adding a distance penalty to the error function being optimized. We believe that changing the pseudo-decoder score, rather than the error minimized, is a novel technique and qualitatively different from other treatments.

Our optimization technique is novel in that it is not a line search method like MERT, nor a (sub)gradient approximation method, nor a simplex method. Rather, it is a regression-based derivative-free trust-region method. Use of regression on scaled weights allows us to take smooth approximations of the error function, which should aid the method's robustness. Trust-region optimization supports the multiresolution placement of regression points, providing a thorough search.

3 Tuner description

We divide the tuner description into three sections. In §3.1 we describe optimization techniques used to optimize efficiently, avoiding local optima. In §3.2 we describe techniques used to make the translations in optimization similar to the output of the decoder. In §3.3 we give techniques used to make the result of tuning robust to human evaluation of test sets.

3.1 Optimization

3.1.1 Scaling

The scalar product (1) used in the determination of the 1-best translation means that the decoder output is scale-invariant. However, many tuning algorithms (excluding MERT and Drem, but including MIRA (Chiang et al., 2008), Ultraconservative Updating (Liu et al., 2012), and others) are impacted by the magnitude of the weight vectors. In this section we show how we rescale all weights and features to change to an intrinsic unit scaling.

Our first step in defining the coordinate system is whitening the feature space, which is transforming the features to be uncorrelated and have equal variance. Whitening the features removes the complications of features with dramatically different scales and features that tend to move nearly in lock-step with each other.

We perform the whitening of the feature space by performing principal component analysis of the matrix M, which we define to be the mean covariance matrix. That is, M is the average of the sample covariance matrices for the different segments, where we consider data from "relatively good" decodes[1].

Principal component analysis of M provides the scaling matrix A, which is used to produce the whitened features $\boldsymbol{\varphi}$:

$$\boldsymbol{\varphi} = A\boldsymbol{f}$$

In order to maintain ordering of the product $\boldsymbol{w}^T\boldsymbol{f}$ under the new scaling of the features, we also rescale the weights via

$$\boldsymbol{\lambda} = \frac{A^{-1}\boldsymbol{w}}{\|A^{-1}\boldsymbol{w}\|}$$

We will use the notational convenience of the implicit transformation between the unscaled variables $\boldsymbol{w}$ and $\boldsymbol{f}$ and the scaled variables $\boldsymbol{\lambda}$ and $\boldsymbol{\varphi}$. The scaling matrix A is constant throughout a Drem run, so this should produce no ambiguity.

With these scaled weights on the unit $(k-1)$-sphere, we can use a standard cosine difference between different weights:

$$\text{dist}(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) = \text{acos}(\boldsymbol{\lambda}_1^T\boldsymbol{\lambda}_2) \tag{2}$$

which implies that all distances between vectors will be between zero and π. This distance function is appealing as a geometrically natural measure of distance between direction vectors.

3.1.2 Derivative-free trust-region optimization

Our tuning process can be summarized as performing the development error minimization

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathcal{E}_{\text{dev}}(\boldsymbol{w}, \mathcal{C}_{\text{dev}}) \tag{3}$$

We choose to perform this optimization using a trust-region method (Conn et al., 2000). We repeatedly solve a problem of the form

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}:\text{dist}(\boldsymbol{\lambda},\boldsymbol{\lambda}_0)<\delta} m_\delta(\boldsymbol{\lambda}, \boldsymbol{\lambda}_0) \tag{4}$$

[1] Defined by the user, and precise definition has little impact. We use metrics scaled like BLEU, and weights are "relatively good" if they give an error within 0.0025 of the best decode's.

where δ is the so-called "trust-region radius", and m_δ (to be defined in §3.1.3) is a simple model approximately equal to $\mathcal{E}_{\text{dev}}$.

When an improvement to $\mathcal{E}_{\text{dev}}$ (or $\mathcal{E}_{\text{dev,robust}}$ in §4) is found via (4), a step is taken in that direction. The trust-region radius is enlarged if the improvement is significant, and maintained or decreased otherwise.

Problem (4) is optimized repeatedly, with different central weights $\boldsymbol{\lambda}_0$ and different trust-region radii δ. Convergence is declared if the trust-region radius becomes small enough or the maximum number of iterations is reached[2].

A new, optimal weight $\boldsymbol{w}$ is the output of each Drem run. It will be used to decode the development corpus, and then Drem will be re-run. Overall convergence is achieved if the Drem output weights converge.

This trust-region method is in stark contrast to MERT's line search methodology on a piecewise continuous error. MERT relies heavily on searching along a line, keeping track of where the one-best translations (and therefore the error) change on that line. MERT's method is designed for a piecewise constant error and would be inapplicable for (4). Both our pseudo-decoder score and development error metrics are continuous.

3.1.3 Error surface modeling via sampling

We choose to evaluate the error function at a few sampled points around the current best weight and fit a quadratic or linear model m_δ to it (Conn et al., 2009).

For a linear model, we choose the evaluation set at the scale δ to be the $2(k-1)+1$ points consisting of the central point $\boldsymbol{\lambda}_0$ of the trust-region and the $2(k-1)$ points found by taking steps of $\pm\delta$ in each of the $k-1$ coordinate directions.

The model of the error is defined as the model $m_\delta(\boldsymbol{\lambda}, \boldsymbol{\lambda}_0)$ that minimizes the squared error between the error evaluations and the model.

Using least-squares regularization to model the error surface completely avoids the issue of needing to approximate a gradient or subgradient of the error function. This is by design and avoids the tendency of local behavior to dominate global behavior, in both computational effort and final result (Conn et al., 2009).

[2]Precise definitions of the many optimization parameters have little impact. Our threshold for the minimum trust-region radius is 0.001, and we allow up to 30 iterations of solving (4) per Drem run.

The local coordinate basis on the unit sphere is arbitrary in the definition of $m_\delta(\boldsymbol{\lambda}, \boldsymbol{\lambda}_0)$. We will use this feature to our advantage, choosing a different random basis every time we perform an optimization iteration. This gives the benefit of function evaluation in many random directions at each step, with negligible cost.

3.2 Pseudo-decoder improvement

We now turn to how we will use the information available to Drem to simulate decoding at a new weight.

3.2.1 Decode score interpolation

For development error, we include a penalty so that a translation will get a lower score ("fade away") as one moves farther from the decode weights that produced that translation. Our pseudo-decoder score is an adjusted version of (1),

$$s_{\text{dev}}(j, t, \boldsymbol{\varphi}, \boldsymbol{\lambda}) = s(\boldsymbol{w}, \boldsymbol{f}) + p(j, t, \boldsymbol{\varphi}, \boldsymbol{\lambda}) \quad (5)$$

where p is the penalty for considering a translation at a weight which is distant from the weights which produced it.

We have freedom in choosing the distance penalty function p. Many optimizers, such as MERT, have no such penalty function, so we can replicate their pseudo-decoders by setting p identically equal to zero. We choose to interpolate instead. That is, the decoder and the pseudo-decoder will produce the same n-best list and scores at that weight (modulo inclusion of translations with infinitely bad scores). In equations, this is

$$p(j, t, \boldsymbol{\varphi}, \boldsymbol{\lambda}^{(m)}) = \begin{cases} 0, & (j, t, \boldsymbol{f}) \in \mathcal{N}(\boldsymbol{w}^{(m)}, \mathcal{C}_{\text{dev}}) \\ -\infty, & \text{otherwise} \end{cases}$$

We give our standard choice for p here. Let $d_{\min}(\boldsymbol{\lambda})$ be the distance of a weight $\boldsymbol{\lambda}$ from the nearest weight where the current segment was decoded:

$$d_{\min}(\boldsymbol{\lambda}) = \min_m \operatorname{dist}(\boldsymbol{\lambda}, \boldsymbol{\lambda}^{(m)})$$

and let $d(j, t, \boldsymbol{\varphi}, \boldsymbol{\lambda})$ be the distance to the nearest weight that produced the given translation $(j, t, \boldsymbol{\varphi})$. We define the maximum distance that can produce a finite score to be a multiple of this minimum distance, $d_{\max} = 1000 d_{\min}$. Then we define the penalty function to be

$$p(j, t, \boldsymbol{\varphi}, \boldsymbol{\lambda}) = \begin{cases} 0, & d = d_{\min} \\ \frac{d_{\min} - d}{d_{\max} - d}, & d_{\min} < d < d_{\max} \\ -\infty, & \text{otherwise} \end{cases}$$

We find Drem's decode score interpolation to be extraordinarily beneficial when n-best list reranking is part of the system. If the ranking from the initial decoder differs substantially from that of the rescorer, we have seen other tuners have difficulty producing translations which are both produced by the first decoder and scored highly by the rescorer.

3.2.2 Tabu search

We, like Foster and Kuhn (2009), feel that early tuning iterations should focus on exploring the space. This helps to develop the psuedo-decoder's knowledge of the decoder's output at various weights. To this end, we have the option of constraining the output of a tuning iteration to be a certain distance from all previous decodes. As in Foster and Kuhn (2009), we reduce the effect in later iterations, to allow convergence. We set this distance to 0.25 for the first twenty iterations of Drem, and zero for the final three iterations.

3.2.3 Historical restarts

We, like Foster and Kuhn (2009), have observed that random restarts are often not valuable for tuning. In Drem this may be due in part to the repeatedly randomized coordinate systems. However, historical restarts can sometimes help recover from an early misstep. The set of starting points will then consist of the given weight and the three prior decode weights with the best development error metric values. If enough distinct historical restarts are not available, random restarts will be added until four distinct starting points are found.

3.2.4 Merging replicates

Our final option in this section is related to the standard practice of running several replicates of the tuning process and choosing to use the weights output by just one of them. Instead of choosing a single replicate's result, we allow the user to merge the n-best lists of all the replicates at some mid-way point of Drem. This improves the knowledge of the pseudo-decoder, allowing Drem to use this information to select its final answer.

We allow ten replicates to proceed for twenty iterations of Drem, then merge their n-best lists and optimize for three further iterations.

3.3 Generalization to test data

We find that the weights found by Drem (and other tuners) do not always generalize well to test data. The proper choices here depend strongly on how the development corpus and evaluation metric differ from the test corpus and evaluation metric.

3.3.1 Error function smoothing

To generalize from a development corpus to an unseen test corpus, we choose to smooth the metric function optimized. We do this by using expected metric scores, as in Smith and Eisner (2006), Och (2003), Cherry and Foster (2012), and Liu et al. (2012). We average the sufficient statistics of the available translations, taking the weight of a translation as $\exp(\alpha s_{\text{dev}}(j, t, \boldsymbol{\varphi}, \boldsymbol{\lambda}))/Z_j$. Here Z_j normalizes the probability of the translations of segment j. The smoothing parameter α can vary, with examples in the literature including $\alpha = 1$ (Cherry and Foster, 2012), $\alpha = 3$ (Liu et al., 2012), and $\alpha = \infty$ (i.e., the standard 1-best score, which would be used if the test set was identical to the dev set). We choose our standard setting of $\alpha = 1$.

3.3.2 Metric choice

The most difficult part of this tuning task may well be choosing a development error to optimize that will give a final result that will match well to human judgment. We choose to maximize a combination of NIST score (NIST Report, 2002), Meteor 1.5 score (Denkowski and Lavie, 2014), and Kendall's τ score (Birch and Osborne, 2011)[3]:

$$0.045 \cdot \text{NIST} + 0.45 \cdot \text{Meteor} + 0.1 \cdot \text{Kendall's } \tau$$

where the weights are chosen based on experience, and we smooth all metrics with $\alpha = 1$. The combined score aims to avoid pitfalls of any individual metric. This metric was developed by performing our own human evaluation of the Czech–English direction and requesting human evaluation from the task organizers for the English–Czech direction.

4 Unused options

Drem has several options that were not necessary for this task, and we give a few of them here.

A quadratic model could be chosen in §3.1.3, where we add the cross-terms to get an evaluation set of $2(k-1)^2+1$ points. For the tuning task, tests showed no improvement in the final result with the quadratic model.

In addition to smoothing in the "depth" of the n-best list, we can also smooth the error spatially.

[3] dev set alignments were created by GIZA++, trained on the supplied training and dev corpora

In §3.1.2, we would replace $\mathcal{E}_{\text{dev}}$ with $\mathcal{E}_{\text{dev,robust}}$, where $\mathcal{E}_{\text{dev,robust}}$ is the average taken over a set of nearby weights. For the tuning task, tests showed no improvement in the final result with this spatial smoothing.

We tested the ability of Drem to handle sparse features, adding a total of 58 nontrivial `TargetWordInsertion` and `SourceWordDeletion` features. Drem ran successfully on this larger feature space, to apparent convergence. However, the resulting translations of the dev set were not qualitatively better, despite the increased risk of overfitting to the dev set.

5 Implementation

The Drem algorithm was programmed and run in GNU Octave 3.6.4 in Scientific Linux. It is designed to be called from the command prompt as a drop-in replacement for the MERT executable that is provided with Moses (Koehn et al., 2007). Additional arguments, such as metric choice, expectation parameter α, quadratic or linear error surface model, etc., can be added to the command line.

Tuning proceeded as described above. For this task the test data are unavailable, so we do not know how the test set differs from the development set. We choose parameters for smoothing and robustification that have generalized well in the past, keeping in mind that we could make better choices (such as paring down the dev set) if we knew how the source text of the test differed from the development text.

Convergence appeared to be achieved in both translation directions.

It is noteworthy that for English–Czech the weight for the feature `TranslationModel0` was tuned to near zero. We restarted the tuning process with it fixed at zero and achieved very similar results.

A comparison of results of the Tuning Task can be found in (WMT, 2015).

6 Discussion

In this paper we have introduced a new method for tuning the weighted linear model which arises in finding a statistical machine translation system. We have created a new, lower-dimensional search space in which all features are uncorrelated and have approximately equal variation. We have created a new method for extrapolating known n-best lists to a new point, effectively reordering its simulated n-best list by penalizing the pseudo-decoder score of less trustworthy translations. Finally, we have employed a new, multi-scale optimization method which avoids approximating derivatives and for robustness smooths the error function and its local approximations.

Several different implementations fit within Drem's framework. This paper presents a batch implementation of Drem. The algorithm requires minor modifications if partial decodes are performed, and this has promise for tuning more efficiently.

References

Alexandra Birch and Miles Osborne. 2011. Reordering metrics for mt. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1027–1035, Portland, Oregon, USA, June. 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, Montréal, Canada, June. 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 2008 Conference on Empirical Methods in Natural Language Processing*, pages 224–233, Honolulu, Hawaii, October. Association for Computational Linguistics.

Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. 2000. *Trust-Region Methods*. Society for Industrial and Applied Mathematics.

Andrew R. Conn, Katya Scheinberg, and Luis N. Vicent. 2009. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics.

Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 376–380, Baltimore, Maryland, USA, June. Association for Computational Linguistics.

Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government. Cleared for public release on 02 Jun 2015. Originator reference number RH-15-114114. Case number 88ABW-2015-2731.

George Foster and Roland Kuhn. 2009. Stabilizing minimum error rate training. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 242–249, Athens, Greece, March. Association for Computational Linguistics.

Michel Galley, Chris Quirk, Colin Cherry, and Kristina Toutanova. 2013. Regularized minimum error rate training. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1948–1959, Seattle, Washington, USA, October. Association for Computational Linguistics.

Kevin Gimpel and Noah A. Smith. 2012. Structured ramp loss minimization for machine translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, Montréal, Canada, June. Association for Computational Linguistics.

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, Edinburgh, Scotland, UK., July. 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.

Lemao Liu, Tiejun Zhao, Taro Watanabe, Hailong Cao, and Conghui Zhu. 2012. Expected error minimization with ultraconservative update for SMT. In *Proceedings of COLING 2012: Posters*, pages 723–732, Mumbai, India, December. The COLING 2012 Organizing Committee.

NIST Report. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. Technical report, National Institute of Standards and Technology.

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

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 787–794, Sydney, Australia, July. Association for Computational Linguistics.

Jun Suzuki, Kevin Duh, and Masaaki Nagata. 2011. Distributed minimum error rate training of smt using particle swarm optimization. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 649–657, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

WMT. 2015. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation (WMT '15)*, Lisbon, Portugal, September. Association for Computational Linguistics.

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