A Simplex Armijo Downhill Algorithm for Optimizing Statistical Machine Translation Decoding Parameters

Bing Zhao
IBM T.J. Watson Research
zhaob@us.ibm.com

Shengyuan Chen
IBM T.J. Watson Research
sychen@us.ibm.com

Abstract

We propose a variation of simplex-downhill algorithm specifically customized for optimizing parameters in statistical machine translation (SMT) decoder for better end-user automatic evaluation metric scores for translations, such as versions of BLEU, TER and mixtures of them. Traditional simplex-downhill has the advantage of derivative-free computations of objective functions, yet still gives satisfactory searching directions in most scenarios. This is suitable for optimizing translation metrics as they are not differentiable in nature. On the other hand, Armijo algorithm usually performs line search efficiently given a searching direction. It is a deep hidden fact that an efficient line search method will change the iterations of simplex, and hence the searching trajectories. We propose to embed the Armijo inexact line search within the simplex-downhill algorithm. We show, in our experiments, the proposed algorithm improves over the widely-applied Minimum Error Rate training algorithm for optimizing machine translation parameters.

1 Introduction

A simple log-linear form is used in SMT systems to combine feature functions designed for identifying good translations, with proper weights. However, we often observe that tuning the weight associated with each feature function is indeed not easy. Starting from a N-Best list generated from a translation decoder, an optimizer, such as Minimum Error Rate (MER) (Och, 2003) training, proposes directions to search for a better *weight-vector* λ to combine feature functions. With a given λ, the N-Best list is re-ranked, and newly selected top-1 hypothesis will be used to compute the final MT evaluation metric score. Due to limited variations in the N-Best list, the nature of ranking, and more importantly, the *non-differentiable* objective functions used for MT (such as BLEU (Papineni et al., 2002)), one often found only local optimal solutions to λ, with no clue to walk out of the riddles.

Automatic evaluation metrics of translations known so far are designed to simulate human judgments of translation qualities especially in the aspects of *fluency* and *adequacy*; they are not differentiable in nature. Simplex-downhill algorithm (Nelder and Mead, 1965) does not require the objective function to be differentiable, and this is well-suited for optimizing such automatic metrics. MER searches *each* dimension independently in a *greedy* fashion, while simplex algorithms consider the movement of *all* the dimensions at the same time via three basic operations: *reflection*, *expansion* and *contraction*, to shrink the simplex iteratively to some local optimal. Practically, as also shown in our experiments, we observe simplex-downhill usually gives better solutions over MER with *random restarts* for both, and reaches the solutions much faster in most of the cases. However, simplex-downhill algorithm is an *unconstrained* algorithm, which does not leverage any domain knowledge in machine translation. Indeed, the objective function used in SMT is shown to be a piece-wise linear problem in (Papineni et al., 1998), and this motivated us to embed an inexact line search with Armijo rules (Armijo, 1966) within a simplex to guide the directions for iterative expansion, reflection and contraction operations. Our proposed modification to the simplex algorithm is an embedded backtracking line search, and the algorithm's convergence (McKinnon, 1999) still holds, though it is configured specially here for optimizing automatic machine translation evaluation metrics.

The remainder of the paper is structured as follow: we briefly introduce the optimization problem in section 2; in section 3, our proposed simplex Armijo downhill algorithm is explained in details; experiments comparing relevant algorithms are in section 4; the conclusions and discussions are given in section 5.

2 Notations

Let $\{(e_{i,k}, \bar{c}_{i,k}, S_{i,k}), k \in [1, K]\}$ be the K-Best list for a given input source sentence f_i in a development dataset containing N sentences. $e_{i,k}$ is a English hypothesis at the rank of k; $\bar{c}_{i,k}$ is a cost vector — a vector of feature function values, with M dimensions: $\bar{c}_{i,k} = (c_{i,k,1}, c_{i,k,2} \ldots c_{i,k,M})$; $S_{i,k}$ is a *sentence-level* translation metric *general counter* (e.g. ngram hits for BLEU, or specific types of errors counted in TER, etc.) for the hypothesis. Let $\bar{\lambda}$ be the weight-vector, so that the cost of $e_{i,k}$ is an inner product: $C(e_{i,k}) = \bar{\lambda} \cdot \bar{c}_{i,k}$. The optimization process is then defined as below:

$$k^*(\text{wrt } i) = \arg\min_k \bar{\lambda} \cdot \bar{c}_{i,k} \quad (1)$$

$$\bar{\lambda}^* = \arg\min_{\bar{\lambda}} \text{Eval}(\sum_{i=1}^{N} S_{i,k^*}), \quad (2)$$

Proceedings of NAACL HLT 2009: Short Papers, pages 21–24,
Boulder, Colorado, June 2009.

where Eval is an evaluation *Error* metric for MT, presuming the *smaller* the better internal to an optimizer; in our case, we decompose BLEU, TER (Snover et al., 2006) and (TER-BLEU)/2.0 into corresponding specific counters for each sentence, cache the intermediate counts in $S_{i,k}$, and compute final corpus-level scores using the sum of all counters; Eqn. 1 is simply a ranking process, with regard to the source sentence i, to select the *top-1* hypothesis, indexed by k^* with the lowest cost $C(e_{i,k^*})$ given current $\bar{\lambda}$; Eqn. 2 is a scoring process of computing the final corpus-level MT metrics via the intermediate counters collected from each top1 hypothesis selected in Eqn. 1. Iteratively, the optimizer picks up an initial guess of $\bar{\lambda}$ using current K-Best list, and reaches a solution $\bar{\lambda}^*$, and then updates the event space with *new* K-Best list generated using a decoder with $\bar{\lambda}^*$; it iterates until there is little change to final scores (a local optimal $\bar{\lambda}^*$ is reached).

3 Simplex Armijo Downhill

We integrate the Armijo line search into the simplex-downhill algorithm in Algorithm 1. We take the *reflection*, *expansion* and *contractions* steps[1] from the simplex-downhill algorithm to find a λ' to form a direction $\lambda' - \lambda_{M+1}$ as the input to the Armijo algorithm, which in turn updates λ' to λ^+ as the input for the next iteration of simplex-downhill algorithm. The combined algorithm iterates until the simplex shrink sufficiently within a predefined threshold. Via Armijo algorithm, we avoid the expensive *shrink* step, and slightly speed up the searching process of simplex-downhill algorithm. Also, the simplex-downhill algorithm usually provides a descend direction to start the Armijo algorithm efficiently. Both algorithms are well known to converge. Moreover, the new algorithm changes the searching path of the traditional simplex-downhill algorithm, and usually leads to better local minimal solutions.

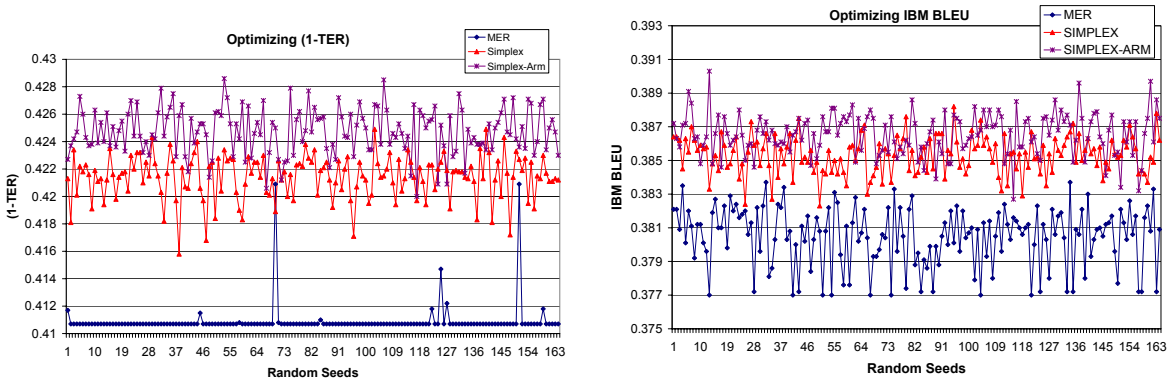
To be more specific, Algorithm 1 clearly conducts an iterative search in the *while* loop from *line* 3 to *line* 28 until the stopping criteria on line 3 is satisfied. Within the loop, the algorithm can be logically divided into two major parts: from line 4 to line 24, it does the simplex-downhill algorithm; the rest does the Armijo search. The simplex-downhill algorithm looks for a lower point by trying the reflection (line 6), expansion (line 10) and contraction (line 17) points in the order showed in the algorithm, which turned out to be very efficient. In rare cases, especially for many dimensions (for instance, 10 to 30 dimensions, as in typical statistical machine translation decoders) none of these three points are not lower enough (line 21), we adapt other means to select lower points. We avoid the traditional expensive shrink procedure, which is not favorable for our machine translation problem neither. Instead we try points on different search lines. Specifically, we test *two* additional points on the line through the highest point and the lowest point, and on the line through the reflection point and the lowest point. It worth pointing out that there are many variants of simplex-downhill algorithm [2], and the implementation described above showed that the algorithm can successfully select a lower λ' in many of our translation test cases to enable the simplex move to a better region of local optimals in the high-dimension space. Our proposed embedded Armijo algorithm, in the second part of the loop (line 25), continues to *refine* the search processes. By backtracking on the segment from λ' to λ_{M+1}, the Armijo algorithm does bring even lower points in our many test cases. With the new lower λ' found by the Armijo algorithm, the simplex-downhill algorithm starts over again. The parameters in line 1 we used are com-

Algorithm 1 Simplex Armijo Downhill Algorithm

1: $\alpha \Leftarrow 1, \gamma \Leftarrow 2, \rho \Leftarrow 0.5, \beta = \eta \Leftarrow 0.9, \epsilon \Leftarrow 1.0 \times 10^{-6}$
2: initilize $(\lambda_1, \cdots, \lambda_{M+1})$
3: **while** $\sum_{i,j=1}^{M+1} \|\lambda_i - \lambda_j\|_2 \leq \epsilon$ **do**
4: sort λ_i ascend
5: $\lambda_o \Leftarrow \frac{1}{N} \sum_{i=1}^{M} \lambda_i$,
6: $\lambda_r \Leftarrow \lambda_o + \alpha(\lambda_o - \lambda_{M+1})$
7: **if** $S(\lambda_1) \leq S(\lambda_r) \leq S(\lambda_M)$ **then**
8: $\lambda' \Leftarrow \lambda_r$
9: **else if** $S(\lambda_r) < S(\lambda_1)$ **then**
10: $\lambda_e \Leftarrow \lambda_o + \gamma(\lambda_o - \lambda_{M+1})$
11: **if** $S(\lambda_e) < S(\lambda_r)$ **then**
12: $\lambda' \Leftarrow \lambda_e$
13: **else**
14: $\lambda' \Leftarrow \lambda_r$
15: **end if**
16: **else if** $S(\lambda_r) > S(\lambda_M)$ **then**
17: $\lambda_c \Leftarrow \lambda_{M+1} + \rho(\lambda_o - \lambda_{M+1})$
18: **if** $S(\lambda_c) < S(\lambda_r)$ **then**
19: $\lambda' \Leftarrow \lambda_c$
20: **else**
21: try points on two additional lines for λ'
22: **end if**
23: **end if**
24: $d \Leftarrow \lambda' - \lambda_{M+1}$
25: $\beta^* \Leftarrow max_{k=0,1,\cdots,40}\{\beta^k | S(\lambda_{M+1} + \beta^k d) - S(\lambda_{M+1}) \leq -\eta \|d\|_2 \beta^k\}$
26: $\lambda^+ = \lambda_{M+1} + \beta^* * d$
27: replace λ_{M+1} with λ^+
28: **end while**

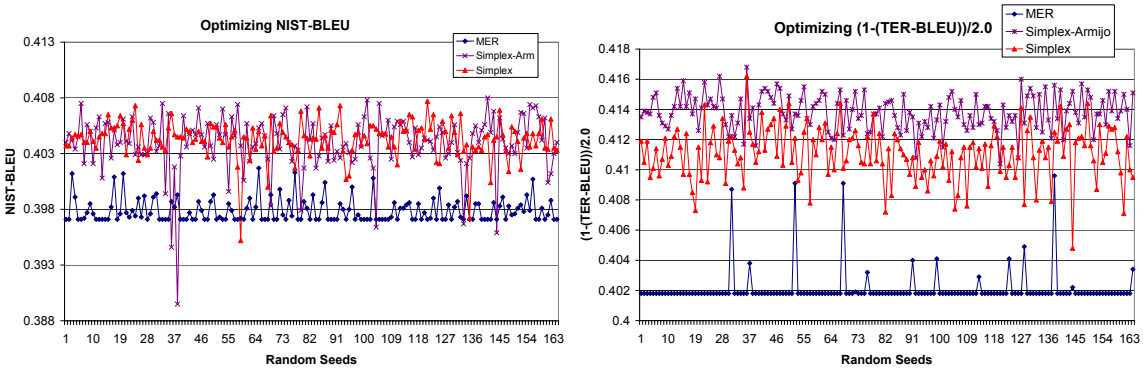
[1] These three basic operations are generally based on heuristics in the traditional simplex-downhill algorithm.

[2] One of such effective tricks for the baseline simplex algorithms can be found here: http://paula.univ.gda.pl/∼dokgrk/simplex.html (link tested to be valid as of 04/03/2009)



(a) Optimizing Toward Metric (1-TER)

(b) Optimizing Toward Metric IBM BLEU



(c) Optimizing Toward Metric NIST BLEU

(d) Optimizing Toward Metric (1-(TER-NISTBLEU))/2

Figure 1: On devset, comparing MER, Simplex Downhill, and Simplex Armijo Downhill Algorithms on different Translation Metrics including TER, IBM BLEU, NIST BLEU, and the combination of TER & NISTBLEU. Empirically, we found optimizing toward (TER-NISTBLEU)/2 gave more reliable solutions on unseen test data. All optimizations are with internal random restarts, and were run from the same 164 random seeds with *multiple iterations* until convergence. Simplex Armijo downhill algorithm is often better than Simplex-downhill algorithm, and is also much better than MER algorithm.

mon ones from literatures and can be tuned further. We find that the combination not only accelerates the searching process to reach similar solutions to the baseline simplex algorithm, but also changes the searching trajectory significantly, leading to even better solutions for machine translation test cases as shown in our experiments.

4 Experiments

Our experiments were carried out on Chinese-English using our syntax-based decoder (Zhao and Al-Onaizan, 2008), a chart-based decoder with tree-to-string [3] grammar, in GALE P3/P3.5 evaluations. There were 10 feature functions computed for each hypothesis, and N-best list size is up to 2,000 per sentence.

Given a weight-vector $\bar{\lambda}_0$, our decoder outputs N-Best *unique* hypotheses for each input source sentence; the event space is then built, and the optimizer is called with a number of random restarts. We used 164 seeds[4] with a small perturbation of three random dimensions in $\bar{\lambda}_0$. The best $\bar{\lambda}_1$ is selected under a given optimizing metric, and is fed back to the decoder to re-generate a *new* N-Best list. Event space is enriched by merging the newly generated N-Best list, and the optimization runs again. This process is iteratively carried out until there are no more improvements observed on a development data set.

We select *three* different metrics: NIST BLEU, IBM BLEU, TER, and a combination of (TER-NISTBLEU)/2 as our optimization goal. On the devset with four references using MT06-NIST text part data, we carried out the optimizations as shown in Figure 1. Over these 164 random restarts in each of the optimizers over the four configurations shown in Figure 1, we found most of the time simplex algorithms perform better than MER in these configurations. Simplex algorithm considers to move all the dimensions at the same time, instead of fixing other

[3]Source shallow constituency tree to target-string rules with variables, forming a probabilistic synchronous context free grammar.

[4]There are 41 servers used in our experiments, four CPUs each.

Table 1: Comparing different optimization algorithms on the held-out speech data, measured on document-average TER, IBMBLEU and (TER-IBMBLEU)/2.0, which were used in GALE P3/3.5 Chinese-English evaluations in Rosetta consortium.

Setup	Broadcast News & Conversation Data		
	BLEUr4n4	TER	(TER-BLEUr4n4)/2
MER	37.36	51.12	6.88
Simplex-Downhill	37.71	50.10	6.19
Simplex Armijo Downhill	38.15	49.92	5.89

dimensions and carrying out a greedy search for one dimension as in MER. With Armijo line search embedded in the simplex-downhill algorithm, the algorithm has a better chance to walk out of the local optimal, via changing the shrinking trajectory of the simplex using a line search to identify the best steps to move. Shown in Figure 1, the solutions from simplex Armijo downhill outperformed the other two under four different optimization metrics for most of the time. Empirically, we found optimizing toward (TER-NISTBLEU)/2 gives marginally better results on final TER and IBM BLEU.

On our devset, we also observed that whenever optimizing toward TER (or mixture of TER & BLEU), MER does not seem to move much, as shown in Figure 1-(a) and Figure 1-(d). However, on BLEU (NIST or IBM version), MER does move reasonably with random restarts. Comparing TER with BLEU, we think the "*shift*" counter in TER is a *confusing* factor to the optimizer, and cannot be computed accurately in the current TER implementations. Also, our random perturbations to the seeds used in restarts might be relatively weaker for MER comparing to our simplex algorithms, though they use exactly the same random seeds. Another fact we found is optimizing toward corpus-level (TER-NISTBLEU)/2 seems to give better performances on most of our unseen datasets, and we choose this as optimization goal to illustrate the algorithms' performances on our unseen testset.

Our test set is the held-out speech part data[5]. We optimize toward corpus-level (TER-NISTBLEU)/2 using devset, and apply the weight-vector on testset to evaluate TER, IBMBLEUr4n4, and a simple combination of (TER-IBMBLEU)/2.0 to compare different algorithms' strengths[6]. Shown in Table 1, simplex Armijo downhill performs the best (though not statistically significant), and the improvements are *consistent* in multiple runs in our observations. Also, given limited resources, such as number of machines and fixed time schedule, both simplex algorithms can run with more random restarts than MER, and can potentially reach better solutions.

[5]Transcriptions of broadcast news and broadcast conversion in MT06; there are 565 sentences, or 11,691 words after segmentation.

[6]We choose document-average metrics to show here simply because they were chosen/required in our GALE P3/P3.5 evaluations for both Arabic-English and Chinese-English individual systems and syscombs.

5 Conclusions and Discussions

We proposed a simplex Armijo downhill algorithm for improved optimization solutions over the standard simplex-downhill and the widely-applied MER. The Armijo algorithm changes the trajectories for the simplex to shrink to a local optimal, and empowers the algorithm a better chance to walk out of the riddled error surface computed by automatic MT evaluation metrics. We showed, empirically, such utilities under several evaluation metrics including BLEU, TER, and a mixture of them. In the future, we plan to integrate domain specific heuristics via approximated derivatives of evaluation metrics or mixture of them to guide the optimizers move toward better solutions for simplex-downhill algorithms.

References

L. Armijo. 1966. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 6:1–3.

K.I.M. McKinnon. 1999. Convergence of the nelder-mead simplex method to a non-stationary point. *SIAM J Optimization*, 9:148–158.

J.A. Nelder and R. Mead. 1965. A simplex method for function minimization. *The Computer Journal*, 7:308–313.

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

Kishore Papineni, Salim Roukos, and Todd Ward. 1998. Maximum likelihood and discriminative training of direct translation models. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech & Signal Processing*, volume 1, pages 189–192, Seattle, May.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. of the 40th Annual Conf. of the Association for Computational Linguistics (ACL 02)*, pages 311–318, Philadelphia, PA, July.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *AMTA*.

Bing Zhao and Yaser Al-Onaizan. 2008. Generalizing local and non-local word-reordering patterns for syntax-based machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.