Learning Non-linear Features for Machine Translation Using Gradient Boosting Machines

Kristina Toutanova
Microsoft Research
Redmond, WA 98502
kristout@microsoft.com

Byung-Gyu Ahn*
Johns Hopkins University
Baltimore, MD 21218
bahn@cs.jhu.edu

Abstract

In this paper we show how to automatically induce non-linear features for machine translation. The new features are selected to approximately maximize a BLEU-related objective and decompose on the level of local phrases, which guarantees that the asymptotic complexity of machine translation decoding does not increase. We achieve this by applying gradient boosting machines (Friedman, 2000) to learn new weak learners (features) in the form of regression trees, using a differentiable loss function related to BLEU. Our results indicate that small gains in performance can be achieved using this method but we do not see the dramatic gains observed using feature induction for other important machine learning tasks.

1 Introduction

The linear model for machine translation (Och and Ney, 2002) has become the de-facto standard in the field. Recently, researchers have proposed a large number of additional features (TaroWatanabe et al., 2007; Chiang et al., 2009) and parameter tuning methods (Chiang et al., 2008b; Hopkins and May, 2011; Cherry and Foster, 2012) which are better able to scale to the larger parameter space. However, a significant feature engineering effort is still required from practitioners. When a linear model does not fit well, researchers are careful to manually add important feature conjunctions, as for example, (Daumé III and Jagarlamudi, 2011; Clark et al., 2012). In the related field of web search ranking, automatically learned non-linear features have brought dramatic improvements in quality (Burges et al., 2005; Wu et al., 2010). Here we adapt the main insights of such work to the machine translation setting and share results on two language pairs.

Some recent works have attempted to relax the linearity assumption on MT features (Nguyen et al., 2007), by defining non-parametric models on complete translation hypotheses, for use in an n-best re-ranking setting. In this paper we develop a framework for inducing non-linear features in the form of regression decision trees, which decompose locally and can be integrated efficiently in decoding. The regression trees encode non-linear feature combinations of the original features. We build on the work by Friedman (2000) which shows how to induce features to minimize any differentiable loss function. In our application the features are regression decision trees, and the loss function is the pairwise ranking log-loss from the PRO method for parameter tuning (Hopkins and May, 2011). Additionally, we show how to design the learning process such that the induced features are local on phrase-pairs and their language model and reordering context, and thus can be incorporated in decoding efficiently.

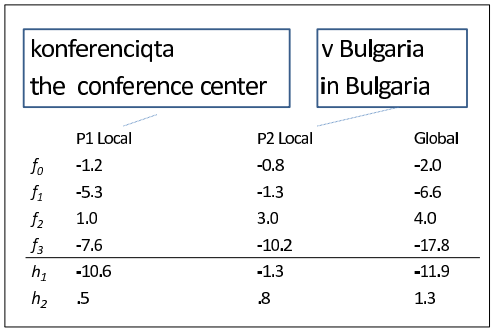
Our results using re-ranking on two language pairs show that the feature induction approach can bring small gains in performance. Overall, even though the method shows some promise, we do not see the dramatic gains that have been seen for the web search ranking task (Wu et al., 2010). Further improvements in the original feature set and the induction algorithm, as well as full integration in decoding are needed to potentially result in substantial performance improvements.

2 Feature learning using gradient boosting machines

In the linear model for machine translation, the scores of translation hypotheses are weighted sums of a set of input features over the hypotheses.

*This research was conducted during the author's internship at Microsoft Research

Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pages 406–411,
Sofia, Bulgaria, August 4-9 2013.



	P1 Local	P2 Local	Global
f_0	-1.2	-0.8	-2.0
f_1	-5.3	-1.3	-6.6
f_2	1.0	3.0	4.0
f_3	-7.6	-10.2	-17.8
h_1	-10.6	-1.3	-11.9
h_2	.5	.8	1.3

Figure 1: A Bulgarian source sentence (meaning "the conference in Bulgaria", together with a candidate translation. Local and global features for the translation hypothesis are shown. f_0=smoothed relative frequency estimate of $\log p(s|t)$; f_1=lexical weighting estimate of $\log p(s|t)$; f_2=joint count of the phrase-pair; f_3=sum of language model log-probabilities of target phrase words given context.

For a set of features $f_1(h), \ldots, f_L(h)$ and weights for these features $\lambda_1, \ldots, \lambda_L$, the hypothesis scores are defined as: $F(h) = \sum_{l=1\ldots L} \lambda_l f_l(h)$. In current state-of-the-art models, the features $f_l(h)$ decompose locally on phrase-pairs (with language model and reordering context) inside the hypotheses. This enables hypothesis recombination during machine translation decoding, leading to faster and more accurate search. As an example, Figure 1 shows a Bulgarian source sentence (spelled phonetically in Latin script) and a candidate translation. Two phrase-pairs are used to compose the translation, and each phrase-pair has a set of local feature function values. A minimal set of four features is shown, for simplicity. We can see that the hypothesis-level (global) feature values are sums of phrase-level (local) feature values. The score of a translation given feature weights λ can be computed either by scoring the phrase-pairs and adding the scores, or by scoring the complete hypothesis by computing its global feature values. The local feature values do look at some limited context outside of a phrase-pair, to compute language model scores and re-ordering scores; therefore we say that the features are defined on phrase-pairs in context.

We start with such a state-of-the-art linear model with decomposable features and show how we can automatically induce additional features. The new features are also locally decomposable, so that the scores of hypotheses can be computed as sums of phrase-level scores. The new local phrase-level features are non-linear combinations of the original phrase-level features.

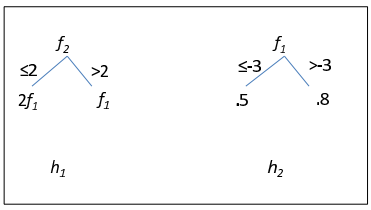


Figure 2: Example of two decision tree features. The left decision tree has linear nodes and the right decision tree has constant nodes.

2.1 Form of induced features

We will use the example in Figure 1 to introduce the form of the new features we induce and to give an intuition of why such features might be useful. The new features are expressed by regression decision trees; Figure 2 shows two examples.

One intuition we might have is that, if a phrase pair has been seen very few times in the training corpus (for example, the first phrase pair P1 in the Figure has been seen only one time $f_2 = 1$), we would like to trust its lexical weighting channel model score f_1 more than its smoothed relative-frequency channel estimate f_0. The first regression tree feature h_1 in Figure 2 captures this intuition. The feature value for a phrase-pair of this feature is computed as follows: if $f_2 \leq 2$, then $h_1(f_0, f_1, f_2, f_3) = 2 \times f_1$; otherwise, $h_1(f_0, f_1, f_2, f_3) = f_1$. The effect of this new feature h_1 is to boost the importance of the lexical weighting score for phrase-pairs of low joint count. More generally, the regression tree features we consider have either linear or constant leaf nodes, and have up to 8 leaves. Deeper trees can capture more complex conditions on several input feature values. Each non-leaf node performs a comparison of some input feature value to a threshold and each leaf node (for linear nodes) returns the value of some input feature multiplied by some factor. For a given regression tree with linear nodes, all leaf nodes are expressions of the same input feature but have different coefficients for it (for example, both leaf nodes of h_1 return affine functions of the input feature f_1). A decision tree feature with constant-valued leaf nodes is illustrated by the right-hand-side tree in Figure 2. For these decision trees, the leaf nodes contain a constant, which is specific to each leaf. These kinds of trees can effectively perform conjunctions of several binary-valued input feature functions; or they can achieve binning of real-values features together with conjunctions over binned values.

Having introduced the form of the new features we learn, we now turn to the methodology for inducing them. We apply the framework of gradient boosting for decision tree weak learners (Friedman, 2000). To define the framework, we need to introduce the original input features, the differentiable loss function, and the details of the tree growing algorithm. We discuss these in turn next.

2.2 Initial features

Our baseline MT system uses relative frequency and lexical weighting channel model weights, one or more language models, distortion penalty, word count, phrase count, and multiple lexicalized reordering weights, one for each distortion type. We have around 15 features in this base feature set. We further expand the input set of features to increase the possibility that useful feature combinations could be found by our feature induction method. The large feature set contains around 190 features, including source and target word count features, joint phrase count, lexical weighting scores according to alternative word-alignment model ran over morphemes instead of words, indicator lexicalized features for insertion and deletion of the top 15 words in each language, cluster-based insertion and deletion indicators using hard word clustering, and cluster based signatures of phrase-pairs. This is the feature set we use as a basis for weak learner induction.

2.3 Loss function

We use a pair-wise ranking log-loss as in the PRO parameter tuning method (Hopkins and May, 2011). The loss is defined by comparing the model scores of pairs of hypotheses h_i and h_j where the BLEU score of the first hypothesis is greater than the BLEU score of the second hypothesis by a specified threshold. [1]

We denote the sentences in a corpus as $s^1, s^2, \ldots, s^N$. For each sentence s^n, we denote the ordered selected pairs of hypotheses as $[h^n_{i_1}, h^n_{j_1}], \ldots, [h^n_{i_K}, h^n_{j_K}]$. The loss-function Ψ is defined in terms of the hypothesis model scores $F(h)$ as follows: $\sum_{n=1\ldots N} \sum_{k=1\ldots K} log(1 + e^{F(h^n_{j_k}) - F(h^n_{i_k})})$.

[1] In our implementation, for each sentence, we sample $10,000$ pairs of translations and accept a pair of translations for use with probability proportional to the BLEU score difference, if that difference is greater than the threshold of 0.04. The top $K = 100$ or $K = 300$ hypothesis pairs with the largest BLEU difference are selected for computation of the loss. We compute sentence-level BLEUscores by add-α smoothing of the match counts for computation of n-gram precision. The α and K parameters are chosen via cross-validation.

1: $F_0(x) = \arg\min_\lambda \Psi(F(x, \lambda))$
2: **for** $m = 1$**to**M **do**
3: $\quad y_r = -[\frac{\partial \Psi(F(x))}{\partial F(x_r)}]_{F(x)=F_{m-1}(x)}$, $r = 1 \ldots R$
4: $\quad \alpha_m = \arg\min_{\alpha,\beta} \sum_{r=1}^{R} [y_r - \beta h(x_i; \alpha)]^2$
5: $\quad \rho_m = \arg\min_\rho \Psi(F_{m-1}(x) + \rho h(x; \alpha_m)$
6: $\quad F_m(x) = F_{m-1}(x) + \rho_m h(x; \alpha_m)$
7: **end for**

Figure 3: A gradient boosting algorithm for local feature functions.

The idea of the gradient boosting method is to induce additional features by computing a functional gradient of the target loss function and iteratively selecting the next weak learner (feature) that is most parallel to the negative gradient. Since we want to induce features such that the hypothesis scores decompose locally, we need to formulate our loss function as a function of local phrase-pair in context scores. Having the model scores decompose locally means that the scores of hypotheses $F(h)$ decompose as $F(h) = \sum_{p_r \in h} F(p_r))$, where by $p_r \in h$ we denote the enumeration over phrase pairs in context that are parts of h. If x_r denotes the input feature vector for a phrase-pair in context p_r, the score of this phrase-pair can be expressed as $F(x_r)$. Appendix A expresses the pairwise log-loss as a function of the phrase scores.

We are now ready to introduce the gradient boosting algorithm, summarized in Figure 3. In the first step of the algorithm, we start by setting the phrase-pair in context scoring function $F_0(x)$ as a linear function of the input feature values, by selecting the feature weights λ to minimize the PRO loss $\Psi(F_0(x))$ as a function of λ. The initial scores have the form $F_0(x) = \sum_{l=1\ldots L} \lambda_l f_l(x)$.This is equivalent to using the (Hopkins and May, 2011) method of parameter tuning for a fixed input feature set and a linear model. We used LBFGS for the optimization in Line 1. Then we iterate and induce a new decision tree weak learner $h(x; \alpha_m)$ like the examples in Figure 2 at each iteration. The parameter vectors α_m encode the topology and parameters of the decision trees, including which feature value is tested at each node, what the comparison cutoffs are, and the way to compute the values at the leaf nodes. After a new decision tree

Language	Train	Dev-Train	Dev-Select	Test
Chs-En	999K	NIST02+03	2K	NIST05
Fin-En	2.2M	12K	2K	4.8K

Table 1: Data sets for the two language pairs Chinese-English and Finnish-English.

		Chs-En		Fin-En	
Features	Tune	Dev-Train	Test	Dev-Train	Test
base	MERT	31.3	30.76	49.8	51.31
base	PRO	31.1	31.16	49.7	51.56
large	PRO	31.8	**31.44**	49.8	51.77
boost-global	PRO	31.8	31.30	50.0	51.87
boost-local	PRO	31.8	**31.44**	50.1	51.95

Table 2: Results for the two language pairs using different weight tuning methods and feature sets.

$h(x;\alpha_m)$ is induced, it is treated as new feature and a linear coefficient ρ_m for that feature is set by minimizing the loss as a function of this parameter (Line 5). The new model scores are set as the old model scores plus a weighted contribution from the new feature (Line 6). At the end of learning, we have a linear model over the input features and additional decision tree features. $F_M(x) = \sum_{l=1\ldots L} \lambda_l f_l(x) + \sum_{m=1\ldots M} \rho_m h(x;\alpha_m)$. The most time-intensive step of the algorithm is the selection of the next decision tree h. This is done by first computing the functional gradient of the loss with respect to the phrase scores $F(x_r)$ at the point of the current model scores $F_{m-1}(x_r)$. Appendix A shows a derivation of this gradient. We then induce a regression tree using mean-square-error minimization, setting the direction given by the negative gradient as a target to be predicted using the features of each phrase-pair in context instance. This is shown as the setting of the α_m parameters by mean-squared-error minimization in Line 4 of the algorithm. The minimization is done approximately by a standard greedy tree-growing algorithm (Breiman et al., 1984). When we tune weights to minimize the loss, such as the weights λ of the initial features, or the weights ρ_m of induced learners, we also include an L_2 penalty on the parameters, to prevent overfitting.

3 Experiments

We report experimental results on two language pairs: Chinese-English, and Finnish-English. Table 1 summarizes statistics about the data. For each language pair, we used a training set (Train) for extracting phrase tables and language models, a Dev-Train set for tuning feature weights and inducing features, a Dev-Select set for selecting hyperparameters of PRO tuning and selecting a stopping point and other hyperparameters of the boosting method, and a Test set for reporting final results. For Chinese-English, the training corpus consists of approximately one million sentence pairs from the FBIS and HongKong portions of the LDC data for the NIST MT evaluation and the Dev-Train and Test sets are from NIST competitions. The MT system is a phrasal system with a 4-gram language model, trained on the Xinhua portion of the English Gigaword corpus. The phrase table has maximum phrase length of 7 words on either side. For Finnish-English we used a dataset from a technical domain of software manuals. For this language pair we used two language models: one very large model trained on billions of words, and another language model trained from the target side of the parallel training set. We report performance using the BLEU-SBP metric proposed in (Chiang et al., 2008a). This is a variant of BLEU (Papineni et al., 2002) with strict brevity penalty, where a long translation for one sentence can not be used to counteract the brevity penalty for another sentence with a short translation. Chiang et al. (2008a) showed that this metric overcomes several undesirable properties of BLEU and has better correlation with human judgements. In our experiments with different feature sets and hyperparameters we observed more stable results and better correlation of Dev-Train, Dev-Select, and Test results using BLEU-SBP. For our experiments, we first trained weights for the base feature sets described in Section 2.2 using MERT. We then decoded the Dev-Train, Dev-Select, and Test datasets, generating 500-best lists for each set. All results in Table 2 report performance of re-ranking on these 500-best lists using different feature sets and parameter tuning methods.

The baseline (base feature set) performance using MERT and PRO tuning on the two language pairs is shown on the first two lines. In line with prior work, PRO tuning achieves a bit lower scores on the tuning set but higher scores on the test set, compared to MERT. The large feature set additionally contains over 170 manually specified features, described in Section 2.2. It was infeasible to run MERT training on this feature set. The test set results using PRO tuning for the large set are about a quarter of a BLEU-SBP point higher than the results using the base feature set on both language pairs. Finally, the last two rows show the performance of the gradient boosting method. In

addition to learning locally decomposable features boost-local, we also implemented boost-global, where we are learning combinations of the global feature values and lose decomposability. The features learned by boost-global can not be computed exactly on partial hypotheses in decoding and thus this method has a speed disadvantage, but we wanted to compare the performance of boost-local and boost-global on n-best list re-ranking to see the potential accuracy gain of the two methods. We see that boost-local is slightly better in performance, in addition to being amenable to efficient decoder integration.

The gradient boosting results are mixed; for Finnish-English, we see around .2 gain of the boost-local model over the large feature set. There is no improvement on Chinese-English, and the boost-global method brings slight degradation. We did not see a large difference in performance among models using different decision tree leaf node types and different maximum numbers of leaf nodes. The selected boost-local model for FIN-ENU used trees with maximum of 2 leaf nodes and linear leaf values; 25 new features were induced before performance started to degrade on the Dev-Select set. The induced features for Finnish included combinations of language model and channel model scores, combinations of word count and channel model scores, and combinations of channel and lexicalized reordering scores. For example, one feature increases the contribution of the relative frequency channel score for phrases with many target words, and decreases the channel model contribution for shorter phrases.

The best boost-local model for Chs-Enu used trees with a maximum of 2 constant-values leaf nodes, and induced 24 new tree features. The features effectively promoted and demoted phrase-pairs in context based on whether an input feature's value was smaller than a determined cutoff.

In conclusion, we proposed a new method to induce feature combinations for machine translation, which do not increase the decoding complexity. There were small improvements on one language pair in a re-ranking setting. Further improvements in the original feature set and the induction algorithm, as well as full integration in decoding are needed to result in substantial performance improvements.

This work did not consider alternative ways of generating non-linear features, such as taking products of two or more input features. It would be interesting to compare such alternatives to the regression tree features we explored.

References

Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. 1984. *Classification and Regression Trees*. Chapman and Hall.

Chris Burges, Tal Shaked, Erin Renshaw, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *ICML*.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *HLT-NAACL*.

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

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

D. Chiang, W. Wang, and K. Knight. 2009. 11,001 new features for statistical machine translation. In *NAACL*.

Jonathan Clark, Alon Lavie, and Chris Dyer. 2012. One system, many domains: Open-domain statistical machine translation via feature augmentation. In *AMTA*.

Hal Daumé III and Jagadeesh Jagarlamudi. 2011. Domain adaptation for machine translation by mining unseen words. In *ACL*.

Jerome H. Friedman. 2000. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *EMNLP*.

Patrick Nguyen, Milind Mahajan, and Xiaodong He. 2007. Training non-parametric features for statistical machine translation. In *Second Workshop on Statistical Machine Translation*.

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

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

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

Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3), June.

4 Appendix A: Derivation of derivatives

Here we express the loss as a function of phrase-level in context scores and derive the derivative of the loss with respect to these scores.

Let us number all phrase-pairs in context in all hypotheses in all sentences as $p_1, \ldots, p_R$ and denote their input feature vectors as $\mathbf{x}_1, \ldots, \mathbf{x}_R$. We will use $F(p_r)$ and $F(x_r)$ interchangeably, because the score of a phrase-pair in context is defined by its input feature vector. The loss $\Psi(F(x_r))$ is expressed as follows: $\sum_{n=1}^{N}\sum_{k=1}^{K} log(1 + e^{\sum_{p_r \in h^n_{j_k}} F(x_r) - \sum_{p_r \in h^n_{i_k}} F(x_r)})$.

Next we derive the derivatives of the loss $\Psi(F(x))$ with respect to the phrase scores. Intuitively, we are treating the scores we want to learn as parameters for the loss function; thus the loss function has a huge number of parameters, one for each instance of each phrase pair in context in each translation. We ask the loss function if these scores could be set in an arbitrary way, what direction it would like to move them in to be minimized. This is the direction given by the negative gradient.

Each phrase-pair in context p_r occurs in exactly one hypothesis h in one sentence. It is possible that two phrase-pairs in context share the same set of input features, but for ease of implementation and exposition, we treat these as different training instances. To express the gradient with respect to $F(x_r)$ we therefore need to focus on the terms of the loss from a single sentence and to take into account the hypothesis pairs $[h_{j,k}, h_{i,k}]$ where the left or the right hypothesis is the hypothesis h containing our focus phrase pair p_r. $\frac{\partial \Psi(F(x))}{\partial F(x_r)}$ is expressed as:

$$= \sum_{k:h=h_{i_k}} -\frac{e^{\sum_{p_r \in h^n_{j_k}} F(x_r) - \sum_{p_r \in h^n_{i_k}} F(x_r)}}{1+e^{\sum_{p_r \in h^n_{j_k}} F(x_r) - \sum_{p_r \in h^n_{i_k}} F(x_r)}}$$

$$+ \sum_{k:h=h_{j_k}} \frac{e^{\sum_{p_r \in h^n_{j_k}} F(x_r) - \sum_{p_r \in h^n_{i_k}} F(x_r)}}{1+e^{\sum_{p_r \in h^n_{j_k}} F(x_r) - \sum_{p_r \in h^n_{i_k}} F(x_r)}}$$

Since in the boosting step we induce a decision tree to fit the negative gradient, we can see that the feature induction algorithm is trying to increase the scores of phrases that occur in better hypotheses (the first hypothesis in each pair), and it increases the scores more if weaker hypotheses have higher advantage; it is also trying to decrease the scores of phrases in weaker hypotheses that are currently receiving high scores.