Approximation Lasso Methods for Language Modeling

Jianfeng Gao
Microsoft Research
One Microsoft Way
Redmond WA 98052 USA
jfgao@microsoft.com

Hisami Suzuki
Microsoft Research
One Microsoft Way
Redmond WA 98052 USA
hisamis@microsoft.com

Bin Yu
Department of Statistics
University of California
Berkeley., CA 94720 U.S.A.
binyu@stat.berkeley.edu

Abstract

Lasso is a regularization method for parameter estimation in linear models. It optimizes the model parameters with respect to a loss function subject to model complexities. This paper explores the use of lasso for statistical language modeling for text input. Owing to the very large number of parameters, directly optimizing the penalized lasso loss function is impossible. Therefore, we investigate two approximation methods, the boosted lasso (BLasso) and the forward stagewise linear regression (FSLR). Both methods, when used with the exponential loss function, bear strong resemblance to the boosting algorithm which has been used as a discriminative training method for language modeling. Evaluations on the task of Japanese text input show that BLasso is able to produce the best approximation to the lasso solution, and leads to a significant improvement, in terms of character error rate, over boosting and the traditional maximum likelihood estimation.

1 Introduction

Language modeling (LM) is fundamental to a wide range of applications. Recently, it has been shown that a linear model estimated using discriminative training methods, such as the boosting and perceptron algorithms, outperforms significantly a traditional word trigram model trained using maximum likelihood estimation (MLE) on several tasks such as speech recognition and Asian language text input (Bacchiani et al. 2004; Roark et al. 2004; Gao et al. 2005; Suzuki and Gao 2005).

The success of discriminative training methods is largely due to fact that unlike the traditional approach (e.g., MLE) that maximizes the function (e.g., likelihood of training data) that is loosely associated with error rate, discriminative training methods aim to directly minimize the error rate on training data even if they reduce the likelihood. However, given a finite set of training samples, discriminative training methods could lead to an arbitrary complex model for the purpose of achieving zero training error. It is well-known that complex models exhibit high variance and perform poorly on unseen data. Therefore some regularization methods have to be used to control the complexity of the model.

Lasso is a regularization method for parameter estimation in linear models. It optimizes the model parameters with respect to a loss function subject to model complexities. The basic idea of lasso is originally proposed by Tibshirani (1996). Recently, there have been several implementations and experiments of lasso on multi-class classification tasks where only a small number of features need to be handled and the lasso solution can be directly computed via numerical methods. To our knowledge, this paper presents the first empirical study of lasso for a realistic, large scale task: LM for Asian language text input. Because the task utilizes millions of features and training samples, directly optimizing the penalized lasso loss function is impossible. Therefore, two approximation methods, the boosted lasso (BLasso, Zhao and Yu 2004) and the forward stagewise linear regression (FSLR, Hastie et al. 2001), are investigated. Both methods, when used with the exponential loss function, bear strong resemblance to the boosting algorithm which has been used as a discriminative training method for LM. Evaluations on the task of Japanese text input show that BLasso is able to produce the best approximation to the lasso solution, and leads to a significant improvement, in terms of character error rate, over the boosting algorithm and the traditional MLE.

2 LM Task and Problem Definition

This paper studies LM on the application of Asian language (e.g. Chinese or Japanese) text input, a standard method of inputting Chinese or Japanese text by converting the input phonetic symbols into the appropriate word string. In this paper we call the task IME, which stands for

input method editor, based on the name of the commonly used Windows-based application.

Performance on IME is measured in terms of the character error rate (CER), which is the number of characters wrongly converted from the phonetic string divided by the number of characters in the correct transcript.

Similar to speech recognition, IME is viewed as a Bayes decision problem. Let A be the input phonetic string. An IME system's task is to choose the most likely word string W^* among those candidates that could be converted from A:

$$W^* = \arg\max_{W \in \mathbf{GEN}(A)} P(W \mid A) = \arg\max_{W \in \mathbf{GEN}(A)} P(W)P(A \mid W) \quad (1)$$

where $\mathbf{GEN}(A)$ denotes the candidate set given A. Unlike speech recognition, however, there is no acoustic ambiguity as the phonetic string is inputted by users. Moreover, we can assume a unique mapping from W and A in IME as words have unique readings, i.e. $P(A \mid W) = 1$. So the decision of Equation (1) depends solely upon $P(W)$, making IME an ideal evaluation test bed for LM.

In this study, the LM task for IME is formulated under the framework of linear models (e.g., Duda et al. 2001). We use the following notation, adapted from Collins and Koo (2005):

- Training data is a set of example input/output pairs. In LM for IME, training samples are represented as $\{A_i, W_i^R\}$, for $i = 1 \ldots M$, where each A_i is an input phonetic string and W_i^R is the reference transcript of A_i.
- We assume some way of generating a set of candidate word strings given A, denoted by $\mathbf{GEN}(A)$. In our experiments, $\mathbf{GEN}(A)$ consists of top n word strings converted from A using a baseline IME system that uses only a word trigram model.
- We assume a set of D+1 features $f_d(W)$, for $d = 0 \ldots D$. The features could be arbitrary functions that map W to real values. Using vector notation, we have $\mathbf{f}(W) \in \Re^{D+1}$, where $\mathbf{f}(W) = [f_0(W), f_1(W), \ldots, f_D(W)]^{\mathrm{T}}$. $f_0(W)$ is called the base feature, and is defined in our case as the log probability that the word trigram model assigns to W. Other features ($f_d(W)$, for $d = 1 \ldots D$) are defined as the counts of word n-grams (n = 1 and 2 in our experiments) in W.
- Finally, the parameters of the model form a vector of D+1 dimensions, each for one feature function, $\boldsymbol{\lambda} = [\lambda_0, \lambda_1, \ldots, \lambda_D]$. The score of a word string W can be written as

$$Score(W, \boldsymbol{\lambda}) = \boldsymbol{\lambda}\mathbf{f}(W) = \sum_{d=0}^{D} \lambda_d f_d(W). \quad (2)$$

The decision rule of Equation (1) is rewritten as

$$W^*(A, \boldsymbol{\lambda}) = \arg\max_{W \in \mathbf{GEN}(A)} Score(W, \boldsymbol{\lambda}). \quad (3)$$

Equation (3) views IME as a ranking problem, where the model gives the ranking score, not probabilities. We therefore do not evaluate the model via perplexity.

Now, assume that we can measure the number of conversion errors in W by comparing it with a reference transcript W^R using an error function $\mathrm{Er}(W^R, W)$, which is the string edit distance function in our case. We call the sum of error counts over the training samples *sample risk*. Our goal then is to search for the best parameter set $\boldsymbol{\lambda}$ which minimizes the sample risk, as in Equation (4):

$$\boldsymbol{\lambda}_{MSR} \overset{def}{=} \arg\min_{\boldsymbol{\lambda}} \sum_{i=1 \ldots M} \mathrm{Er}(W_i^R, W_i^*(A_i, \boldsymbol{\lambda})). \quad (4)$$

However, (4) cannot be optimized easily since Er(.) is a piecewise constant (or step) function of $\boldsymbol{\lambda}$ and its gradient is undefined. Therefore, discriminative methods apply different approaches that optimize it approximately. The boosting algorithm described below is one of such approaches.

3 Boosting

This section gives a brief review of the boosting algorithm, following the description of some recent work (e.g., Schapire and Singer 1999; Collins and Koo 2005).

The boosting algorithm uses an exponential loss function (ExpLoss) to approximate the sample risk in Equation (4). We define the margin of the pair (W^R, W) with respect to the model $\boldsymbol{\lambda}$ as

$$M(W^R, W) = Score(W^R, \boldsymbol{\lambda}) - Score(W, \boldsymbol{\lambda}) \quad (5)$$

Then, ExpLoss is defined as

$$\mathrm{ExpLoss}(\boldsymbol{\lambda}) = \sum_{i=1 \ldots M} \sum_{W_i \in \mathbf{GEN}(A_i)} \exp(-M(W_i^R, W_i)) \quad (6)$$

Notice that ExpLoss is convex so there is no problem with local minima when optimizing it. It is shown in Freund et al. (1998) and Collins and Koo (2005) that there exist gradient search procedures that converge to the right solution.

Figure 1 summarizes the boosting algorithm we used. After initialization, Steps 2 and 3 are

1 Set $\lambda_0 = \arg\min_{\lambda_0} \mathrm{ExpLoss}(\boldsymbol{\lambda})$; and $\lambda_d = 0$ for d=1…D
2 Select a feature f_{k^*} which has largest estimated impact on reducing ExpLoss of Eq. (6)
3 Update $\lambda_{k^*} \leftarrow \lambda_{k^*} + \delta^*$, and return to Step 2

Figure 1: The boosting algorithm

repeated N times; at each iteration, a feature is chosen and its weight is updated as follows.

First, we define Upd($\boldsymbol{\lambda}$, k, δ) as an updated model, with the same parameter values as $\boldsymbol{\lambda}$ with the exception of λ_k, which is incremented by δ

$$\mathrm{Upd}(\boldsymbol{\lambda}, k, \delta) = \{\lambda_0, \lambda_1, ..., \lambda_k + \delta, ..., \lambda_D\}$$

Then, Steps 2 and 3 in Figure 1 can be rewritten as Equations (7) and (8), respectively.

$$(k^*, \delta^*) = \arg\min_{k,\delta} \mathrm{ExpLoss}(\mathrm{Upd}(\boldsymbol{\lambda}, k, \delta)) \quad (7)$$

$$\boldsymbol{\lambda}^t = \mathrm{Upd}(\boldsymbol{\lambda}^{t-1}, k^*, \delta^*) \quad (8)$$

The boosting algorithm can be too greedy: Each iteration usually reduces the ExpLoss(.) on training data, so for the number of iterations large enough this loss can be made arbitrarily small. However, fitting training data too well eventually leads to overfiting, which degrades the performance on unseen test data (even though in boosting overfitting can happen very slowly).

Shrinkage is a simple approach to dealing with the overfitting problem. It scales the incremental step δ by a small constant ν, $\nu \in (0, 1)$. Thus, the update of Equation (8) with shrinkage is

$$\boldsymbol{\lambda}^t = \mathrm{Upd}(\boldsymbol{\lambda}^{t-1}, k^*, \nu\delta^*) \quad (9)$$

Empirically, it has been found that smaller values of ν lead to smaller numbers of test errors.

4 Lasso

Lasso is a regularization method for estimation in linear models (Tibshirani 1996). It regularizes or shrinks a fitted model through an L_1 penalty or constraint.

Let $T(\boldsymbol{\lambda})$ denote the L_1 penalty of the model, i.e., $T(\boldsymbol{\lambda}) = \sum_{d=0...D} |\lambda_d|$. We then optimize the model $\boldsymbol{\lambda}$ so as to minimize a regularized loss function on training data, called *lasso loss* defined as

$$\mathrm{LassoLoss}(\boldsymbol{\lambda}, \alpha) = \mathrm{ExpLoss}(\boldsymbol{\lambda}) + \alpha T(\boldsymbol{\lambda}) \quad (10)$$

where $T(\boldsymbol{\lambda})$ generally penalizes larger models (or complex models), and the parameter α controls the amount of regularization applied to the estimate. Setting $\alpha = 0$ reverses the LassoLoss to the unregularized ExpLoss; as α increases, the model coefficients all shrink, each ultimately becoming zero. In practice, α should be adaptively chosen to minimize an estimate of expected loss, e.g., α decreases with the increase of the number of iterations.

Computation of the solution to the lasso problem has been studied for special loss functions. For least square regression, there is a fast algorithm LARS to find the whole lasso path for different α' s (Obsborn et al. 2000a; 2000b; Efron et al. 2004); for 1-norm SVM, it can be transformed into a linear programming problem with a fast algorithm similar to LARS (Zhu et al. 2003). However, the solution to the lasso problem for a general convex loss function and an adaptive α remains open. More importantly for our purposes, directly minimizing lasso function of Equation (10) with respect to $\boldsymbol{\lambda}$ is not possible when a very large number of model parameters are employed, as in our task of LM for IME. Therefore we investigate below two methods that closely approximate the effect of the lasso, and are very similar to the boosting algorithm.

It is also worth noting the difference between L_1 and L_2 penalty. The classical Ridge Regression setting uses an L_2 penalty in Equation (10) i.e., $T(\boldsymbol{\lambda}) = \sum_{d=0...D}(\lambda_d)^2$, which is much easier to minimize (for least square loss but not for ExpLoss). However, recent research (Donoho et al. 1995) shows that the L_1 penalty is better suited for sparse situations, where there are only a small number of features with nonzero weights among all candidate features. We find that our task is indeed a sparse situation: among 860,000 features, in the resulting linear model only around 5,000 features have nonzero weights. We then focus on the L_1 penalty. We leave the empirical comparison of the L_1 and L_2 penalty on the LM task to future work.

4.1 Forward Stagewise Linear Regression (FSLR)

The first approximation method we used is FSLR, described in (Algorithm 10.4, Hastie et al. 2001), where Steps 2 and 3 in Figure 1 are performed according to Equations (7) and (11), respectively.

$$(k^*, \delta^*) = \arg\min_{k,\delta} \mathrm{ExpLoss}(\mathrm{Upd}(\boldsymbol{\lambda}, k, \delta)) \quad (7)$$

$$\boldsymbol{\lambda}^t = \mathrm{Upd}(\boldsymbol{\lambda}^{t-1}, k^*, \varepsilon \times \mathrm{sign}(\delta^*)) \quad (11)$$

Notice that FSLR is very similar to the boosting algorithm with shrinkage in that at each step, the feature f_{k^*} that has largest estimated impact on reducing ExpLoss is selected. The only difference is that FSLR updates the weight of f_{k^*} by a small fixed step size ε. By taking such small steps, FSLR imposes some implicit regularization, and can closely approximate the effect of the lasso in a local sense (Hastie et al. 2001). Empirically, we find that the performance of the boosting algorithm with shrinkage closely resembles that of FSLR, with the learning rate parameter ν corresponding to ε.

4.2 Boosted Lasso (BLasso)

The second method we used is a modified version of the BLasso algorithm described in Zhao and Yu (2004). There are two major differences between BLasso and FSLR. At each iteration, BLasso can take either a *forward step* or a *backward step*. Similar to the boosting algorithm and FSLR, at each forward step, a feature is selected and its weight is updated according to Equations (12) and (13).

$$(k^*, \delta^*) = \arg\min_{k, \delta = \pm\varepsilon} \text{ExpLoss}(\text{Upd}(\boldsymbol{\lambda}, k, \delta)) \quad (12)$$

$$\boldsymbol{\lambda}^t = \text{Upd}(\boldsymbol{\lambda}^{t-1}, k^*, \varepsilon \times \text{sign}(\delta^*)) \quad (13)$$

However, there is an important difference between Equations (12) and (7). In the boosting algorithm with shrinkage and FSLR, as shown in Equation (7), a feature is selected by its impact on reducing the loss with its optimal update δ^*. In contract, in BLasso, as shown in Equation (12), the optimization over δ is removed, and for each feature, its loss is calculated with an update of either $+\varepsilon$ or $-\varepsilon$, i.e., the grid search is used for feature selection. We will show later that this seemingly trivial difference brings a significant improvement.

The backward step is unique to BLasso. In each iteration, a feature is selected and its weight is updated backward if and only if it leads to a decrease of the lasso loss, as shown in Equations (14) and (15):

$$k^* = \arg\min_{k, \lambda_k \neq 0} \text{ExpLoss}(\text{Upd}(\boldsymbol{\lambda}, k, -\text{sign}(\lambda_k) \times \varepsilon) \quad (14)$$

$$\boldsymbol{\lambda}^t = \text{Upd}(\boldsymbol{\lambda}^{t-1}, k^*, -\text{sign}(\lambda_{k^*}) \times \varepsilon) \quad (15)$$

$$\text{if } \text{LassoLoss}(\boldsymbol{\lambda}^{t-1}, \alpha^{t-1}) - \text{LassoLoss}(\boldsymbol{\lambda}^t, \alpha^t) > \theta$$

where θ is a tolerance parameter.

Figure 2 summarizes the BLasso algorithm we used. After initialization, Steps 4 and 5 are repeated N times; at each iteration, a feature is chosen and its weight is updated either backward or forward by a fixed amount ε. Notice that the value of α is adaptively chosen according to the reduction of ExpLoss during training. The algorithm starts with a large initial α, and then at each forward step the value of α decreases until the ExpLoss stops decreasing. This is intuitively desirable: It is expected that most highly effective features are selected in early stages of training, so the reduction of ExpLoss at each step in early stages are more substantial than in later stages. These early steps coincide with the boosting steps most of the time. In other words, the effect of backward steps is more visible at later stages.

Our implementation of BLasso differs slightly from the original algorithm described in Zhao and Yu (2004). Firstly, because the value of the base feature f_0 is the log probability (assigned by a word trigram model) and has a different range from that of other features as in Equation (2), λ_0 is set to optimize ExpLoss in the initialization step (Step 1 in Figure 2) and remains fixed during training. As suggested by Collins and Koo (2005), this ensures that the contribution of the log-likelihood feature f_0 is well-calibrated with respect to ExpLoss. Secondly, when updating a feature weight, if the size of the optimal update step (computed via Equation (7)) is smaller than ε, we use the optimal step to update the feature. Therefore, in our implementation BLasso does not always take a fixed step; it may take steps whose size is smaller than ε. In our initial experiments we found that both changes (also used in our implementations of boosting and FSLR) were crucial to the performance of the methods.

1 Initialize $\boldsymbol{\lambda}^0$: set $\lambda_0 = \text{argmin}_{\lambda 0}\text{ExpLoss}(\boldsymbol{\lambda})$, and $\lambda_d = 0$ for d=1…D.
2 Take a forward step according to Eq. (12) and (13), and the updated model is denoted by $\boldsymbol{\lambda}^1$
3 Initialize $\alpha = (\text{ExpLoss}(\boldsymbol{\lambda}^0) - \text{ExpLoss}(\boldsymbol{\lambda}^1))/\varepsilon$
4 Take a backward step if and only if it leads to a decrease of LassoLoss according to Eq. (14) and (15), where $\theta = 0$; otherwise
5 Take a forward step according to Eq. (12) and (13); update $\alpha = \min(\alpha, (\text{ExpLoss}(\boldsymbol{\lambda}^{t-1}) - \text{ExpLoss}(\boldsymbol{\lambda}^t))/\varepsilon)$; and return to Step 4.

Figure 2: The BLasso algorithm

(Zhao and Yu 2004) provides theoretical justifications for BLasso. It has been proved that (1) it guarantees that it is safe for BLasso to start with an initial α which is the largest α that would allow an ε step away from 0 (i.e., larger α's correspond to $T(\boldsymbol{\lambda})$=0); (2) for each value of α, BLasso performs coordinate descent (i.e., reduces ExpLoss by updating the weight of a feature) until there is no descent step; and (3) for each step where the value of α decreases, it guarantees that the lasso loss is reduced. As a result, it can be proved that for a finite number of features and θ = 0, the BLasso algorithm shown in Figure 2 converges to the lasso solution when $\varepsilon \rightarrow 0$.

5 Evaluation

5.1 Settings

We evaluated the training methods described above in the so-called cross-domain language model adaptation paradigm, where we adapt a model trained on one domain (which we call the

background domain) to a different domain (*adaptation* domain), for which only a small amount of training data is available.

The data sets we used in our experiments came from five distinct sources of text. A 36-million-word Nikkei Newspaper corpus was used as the background domain, on which the word trigram model was trained. We used four adaptation domains: Yomiuri (newspaper corpus), TuneUp (balanced corpus containing newspapers and other sources of text), Encarta (encyclopedia) and Shincho (collection of novels). All corpora have been pre-word-segmented using a lexicon containing 167,107 entries. For each of the four domains, we created training data consisting of 72K sentences (0.9M~1.7M words) and test data of 5K sentences (65K~120K words) from each adaptation domain. The first 800 and 8,000 sentences of each adaptation training data were also used to show how different sizes of training data affected the performances of various adaptation methods. Another 5K-sentence subset was used as held-out data for each domain.

We created the training samples for discriminative learning as follows. For each phonetic string A in adaptation training data, we produced a lattice of candidate word strings W using the baseline system described in (Gao et al. 2002), which uses a word trigram model trained via MLE on the Nikkei Newspaper corpus. For efficiency, we kept only the best 20 hypotheses in its candidate conversion set **GEN**(A) for each training sample for discriminative training. The oracle best hypothesis, which gives the minimum number of errors, was used as the reference transcript of A.

We used unigrams and bigrams that occurred more than once in the training set as features in the linear model of Equation (2). The total number of candidate features we used was around 860,000.

5.2 Main Results

Table 1 summarizes the results of various model training (adaptation) methods in terms of CER (%) and CER reduction (in parentheses) over comparing models. In the first column, the numbers in parentheses next to the domain name indicates the number of training sentences used for adaptation.

Baseline, with results shown in Column 3, is the word trigram model. As expected, the CER correlates very well the similarity between the background domain and the adaptation domain, where domain similarity is measured in terms of cross entropy (Yuan et al. 2005) as shown in Column 2.

MAP (maximum *a posteriori*), with results shown in Column 4, is a traditional LM adaptation method where the parameters of the background model are adjusted in such a way that maximizes the likelihood of the adaptation data. Our implementation takes the form of linear interpolation as described in Bacchiani et al. (2004): $P(w_i|h) = \lambda P_b(w_i|h) + (1-\lambda)P_a(w_i|h)$, where P_b is the probability of the background model, P_a is the probability trained on adaptation data using MLE and the history h corresponds to two preceding words (i.e. P_b and P_a are trigram probabilities). λ is the interpolation weight optimized on held-out data.

Boosting, with results shown in Column 5, is the algorithm described in Figure 1. In our implementation, we use the shrinkage method suggested by Schapire and Singer (1999) and Collins and Koo (2005). At each iteration, we used the following update for the kth feature

$$\delta_k = \frac{1}{2}\log\frac{C_k^+ + \varepsilon Z}{C_k^- + \varepsilon Z} \tag{16}$$

where C_k^+ is a value increasing exponentially with the sum of margins of (W^R, W) pairs over the set where f_k is seen in W^R but not in W; C_k^- is the value related to the sum of margins over the set where f_k is seen in W but not in W^R. ε is a smoothing factor (whose value is optimized on held-out data) and Z is a normalization constant (whose value is the ExpLoss(.) of training data according to the current model). We see that εZ in Equation (16) plays the same role as ν in Equation (9).

BLasso, with results shown in Column 6, is the algorithm described in Figure 2. We find that the performance of BLasso is not very sensitive to the selection of the step size ε across training sets of different domains and sizes. Although small ε is preferred in theory as discussed earlier, it would lead to a very slow convergence. Therefore, in our experiments, we always use a large step ($\varepsilon = 0.5$) and use the so-called early stopping strategy, i.e., the number of iterations before stopping is optimized on held-out data.

In the task of LM for IME, there are millions of features and training samples, forming an extremely large and sparse matrix. We therefore applied the techniques described in Collins and Koo (2005) to speed up the training procedure. The resulting algorithms run in around 15 and 30 minutes respectively for Boosting and BLasso to converge on an XEON™ MP 1.90GHz machine when training on an 8K-sentnece training set.

The results in Table 1 give rise to several observations. First of all, both discriminative training methods (i.e., Boosting and BLasso) outperform MAP substantially. The improvement margins are larger when the background and adaptation domains are more similar. The phenomenon is attributed to the underlying difference between the two adaptation methods: MAP aims to improve the likelihood of a distribution, so if the adaptation domain is very similar to the background domain, the difference between the two underlying distributions is so small that MAP cannot adjust the model effectively. Discriminative methods, on the other hand, do not have this limitation for they aim to reduce errors directly. Secondly, BLasso outperforms Boosting significantly (p-value < 0.01) on all test sets. The improvement margins vary with the training sets of different domains and sizes. In general, in cases where the adaptation domain is less similar to the background domain and larger training set is used, the improvement of BLasso is more visible.

Note that the CER results of FSLR are not included in Table 1 because it achieves very similar results to the boosting algorithm with shrinkage if the controlling parameters of both algorithms are optimized via cross-validation. We shall discuss their difference in the next section.

5.3 Dicussion

This section investigates what components of BLasso bring the improvement over Boosting. Comparing the algorithms in Figures 1 and 2, we notice three differences between BLasso and Boosting: (i) the use of backward steps in BLasso; (ii) BLasso uses the grid search (fixed step size) for feature selection in Equation (12) while Boosting uses the continuous search (optimal step size) in Equation (7); and (iii) BLasso uses a fixed step size for feature update in Equation (13) while Boosting uses an optimal step size in Equation (8). We then investigate these differences in turn.

To study the impact of backward steps, we compared BLasso with the boosting algorithm with a fixed step search and a fixed step update, henceforth referred to as **F-Boosting**. F-Boosting was implemented as Figure 2, by setting a large value to θ in Equation (15), i.e., $\theta = 10^3$, to prohibit backward steps. We find that although the training error curves of BLasso and F-Boosting are almost identical, the $T(\boldsymbol{\lambda})$ curves grow apart with iterations, as shown in Figure 3. The results show that with backward steps, BLasso achieves a better approximation to the true lasso solution: It leads to a model with similar training errors but less complex (in terms of L_1 penalty). In our experiments we find that the benefit of using backward steps is only visible in later iterations when BLasso's backward steps kick in. A typical example is shown in Figure 4. The early steps fit to highly effective features and in these steps BLasso and F-Boosting agree. For later steps, fine-tuning of features is required. BLasso with backward steps provides a better mechanism than F-Boosting to revise the previously chosen features to accommodate this fine level of tuning. Consequently we observe the superior performance of BLasso at later stages as shown in our experiments.

As well-known in linear regression models, when there are many strongly correlated features, model parameters can be poorly estimated and exhibit high variance. By imposing a model size constraint, as in lasso, this phenomenon is alleviated. Therefore, we speculate that a better approximation to lasso, as BLasso with backward steps, would be superior in eliminating the negative effect of strongly correlated features in model estimation. To verify our speculation, we performed the following experiments. For each training set, in addition to word unigram and bigram features, we introduced a new type of features called *headword bigram*.

As described in Gao et al. (2002), headwords are defined as the content words of the sentence. Therefore, headword bigrams constitute a special type of skipping bigrams which can capture dependency between two words that may not be adjacent. In reality, a large portion of headword bigrams are identical to word bigrams, as two headwords can occur next to each other in text. In the adaptation test data we used, we find that headword bigram features are for the most part either *completely overlapping* with the word bigram features (i.e., all instances of headword bigrams also count as word bigrams) or *not overlapping at all* (i.e., a headword bigram feature is not observed as a word bigram feature) – less than 20% of headword bigram features displayed a variable degree of overlap with word bigram features. In our data, the rate of completely overlapping features is 25% to 47% depending on the adaptation domain. From this, we can say that the headword bigram features show moderate to high degree of correlation with the word bigram features.

We then used BLasso and F-Boosting to train the linear language models including both word bigram and headword bigram features. We find that although the CER reduction by adding

headword features is overall very small, the difference between the two versions of BLasso is more visible in all four test sets. Comparing Figures 5 – 8 with Figure 4, it can be seen that BLasso with backward steps outperforms the one without backward steps in much earlier stages of training with a larger margin. For example, on Encarta data sets, BLasso outperforms F-Boosting after around 18,000 iterations with headword features (Figure 7), as opposed to 25,000 iterations without headword features (Figure 4). The results seem to corroborate our speculation that BLasso is more robust in the presence of highly correlated features.

To investigate the impact of using the grid search (fixed step size) versus the continuous search (optimal step size) for feature selection, we compared F-Boosting with FSLR since they differs only in their search methods for feature selection. As shown in Figures 5 to 8, although FSLR is robust in that its test errors do not increase after many iterations, F-Boosting can reach a much lower error rate on three out of four test sets. Therefore, in the task of LM for IME where CER is the most important metric, the grid search for feature selection is more desirable.

To investigate the impact of using a fixed versus an optimal step size for feature update, we compared FSLR with Boosting. Although both algorithms achieve very similar CER results, the performance of FSLR is much less sensitive to the selected fixed step size. For example, we can select any value from 0.2 to 0.8, and in most settings FSLR achieves the very similar lowest CER after 20,000 iterations, and will stay there for many iterations. In contrast, in Boosting, the optimal value of ε in Equation (16) varies with the sizes and domains of training data, and has to be tuned carefully. We thus conclude that in our task FSLR is more robust against different training settings and a fixed step size for feature update is more preferred.

6 Conclusion

This paper investigates two approximation lasso methods for LM applied to a realistic task with a very large number of features with sparse feature space. Our results on Japanese text input are promising. BLasso outperforms the boosting algorithm significantly in terms of CER reduction on all experimental settings.

We have shown that this superior performance is a consequence of BLasso's backward step and its fixed step size in both feature selection and feature weight update. Our experimental results in Section 5 show that the use of backward step is vital for model fine-tuning after major features are selected and for coping with strongly correlated features; the fixed step size of BLasso is responsible for the improvement of CER and the robustness of the results. Experiments on other data sets and theoretical analysis are needed to further support our findings in this paper.

References

Bacchiani, M., Roark, B., and Saraclar, M. 2004. Language model adaptation with MAP estimation and the perceptron algorithm. In *HLT-NAACL 2004*. 21-24.

Collins, Michael and Terry Koo 2005. Discriminative reranking for natural language parsing. *Computational Linguistics* 31(1): 25-69.

Duda, Richard O, Hart, Peter E. and Stork, David G. 2001. *Pattern classification.* John Wiley & Sons, Inc.

Donoho, D., I. Johnstone, G. Kerkyachairan, and D. Picard. 1995. Wavelet shrinkage; asymptopia? (with discussion), *J. Royal. Statist. Soc.* 57: 201-337.

Efron, B., T. Hastie, I. Johnstone, and R. Tibshirani. 2004. Least angle regression. *Ann. Statist.* 32, 407-499.

Freund, Y, R. Iyer, R. E. Schapire, and Y. Singer. 1998. An efficient boosting algorithm for combining preferences. In *ICML'98.*

Hastie, T., R. Tibshirani and J. Friedman. 2001. *The elements of statistical learning.* Springer-Verlag, New York.

Gao, Jianfeng, Hisami Suzuki and Yang Wen. 2002. Exploiting headword dependency and predictive clustering for language modeling. In *EMNLP 2002.*

Gao. J., Yu, H., Yuan, W., and Xu, P. 2005. Minimum sample risk methods for language modeling. In *HLT/EMNLP 2005*.

Osborne, M.R. and Presnell, B. and Turlach B.A. 2000a. A new approach to variable selection in least squares problems. *Journal of Numerical Analysis*, 20(3).

Osborne, M.R. and Presnell, B. and Turlach B.A. 2000b. On the lasso and its dual. *Journal of Computational and Graphical Statistics*, 9(2): 319-337.

Roark, Brian, Murat Saraclar and Michael Collins. 2004. Corrective language modeling for large vocabulary ASR with the perceptron algorithm. In *ICASSP 2004*.

Schapire, Robert E. and Yoram Singer. 1999. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3): 297-336.

Suzuki, Hisami and Jianfeng Gao. 2005. A comparative study on language model adaptation using new evaluation metrics. In *HLT/EMNLP 2005*.

Tibshirani, R. 1996. Regression shrinkage and selection via the lasso. *J. R. Statist. Soc. B*, 58(1): 267-288.

Yuan, W., J. Gao and H. Suzuki. 2005. An Empirical Study on Language Model Adaptation Using a Metric of Domain Similarity. In *IJCNLP 05*.

Zhao, P. and B. Yu. 2004. Boosted lasso. *Tech Report*, Statistics Department, U. C. Berkeley.

Zhu, J. S. Rosset, T. Hastie, and R. Tibshirani. 2003. 1-norm support vector machines. *NIPS 16*. MIT Press.

Table 1. CER (%) and CER reduction (%) (Y=Yomiuri; T=TuneUp; E=Encarta; S=-Shincho)

Domain	Entropy vs.Nikkei	Baseline	MAP (over Baseline)	Boosting (over MAP)	BLasso (over MAP/Boosting)
Y (800)	7.69	3.70	3.70 (+0.00)	3.13 (+15.41)	3.01 (+18.65/+3.83)
Y (8K)	7.69	3.70	3.69 (+0.27)	2.88 (+21.95)	2.85 (+22.76/+1.04)
Y (72K)	7.69	3.70	3.69 (+0.27)	2.78 (+24.66)	2.73 (+26.02/+1.80)
T (800)	7.95	5.81	5.81 (+0.00)	5.69 (+2.07)	5.63 (+3.10/+1.05)
T (8K)	7.95	5.81	5.70 (+1.89)	5.48 (+5.48)	5.33 (+6.49/+2.74)
T (72K)	7.95	5.81	5.47 (+5.85)	5.33 (+2.56)	5.05 (+7.68/+5.25)
E (800)	9.30	10.24	9.60 (+6.25)	9.82 (-2.29)	9.18 (+4.38/+6.52)
E (8K)	9.30	10.24	8.64 (+15.63)	8.54 (+1.16)	8.04 (+6.94/+5.85)
E (72K)	9.30	10.24	7.98 (+22.07)	7.53 (+5.64)	7.20 (+9.77/+4.38)
S (800)	9.40	12.18	11.86 (+2.63)	11.91 (-0.42)	11.79 (+0.59/+1.01)
S (8K)	9.40	12.18	11.15 (+8.46)	11.09 (+0.54)	10.73 (+3.77/+3.25)
S (72K)	9.40	12.18	10.76 (+11.66)	10.25 (+4.74)	9.64 (+10.41/+5.95)

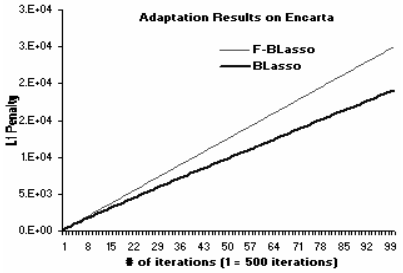

Figure 3. L_1 curves: models are trained on the E(8K) dataset.

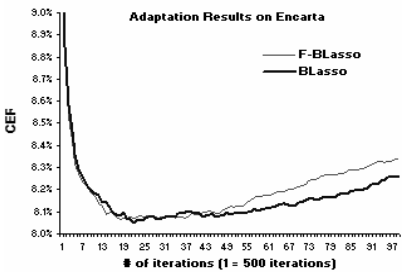

Figure 4. Test error curves: models are trained on the E(8K) dataset.

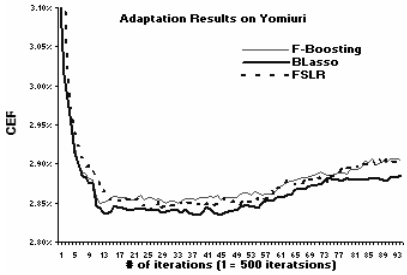

Figure 5. Test error curves: models are trained on the Y(8K) dataset, including headword bigram features.

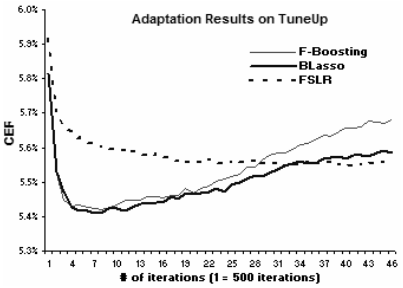

Figure 6. Test error curves: models are trained on the T(8K) dataset, including headword bigram features.

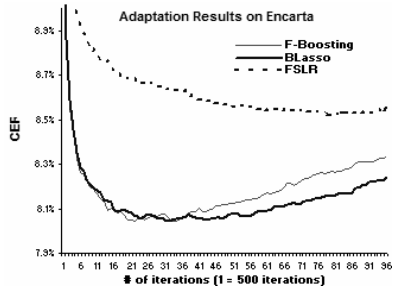

Figure 7. Test error curves: models are trained on the E(8K) dataset, including headword bigram features.

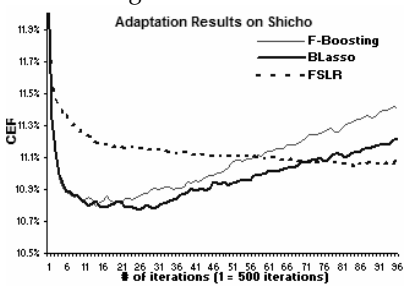

Figure 8. Test error curves: models are trained on the S(8K) dataset, including headword bigram features.