Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty

Yoshimasa Tsuruoka[†‡] **Jun'ichi Tsujii**[†‡*] **Sophia Ananiadou**[†‡]
[†] School of Computer Science, University of Manchester, UK
[‡] National Centre for Text Mining (NaCTeM), UK
[*] Department of Computer Science, University of Tokyo, Japan
{yoshimasa.tsuruoka,j.tsujii,sophia.ananiadou}@manchester.ac.uk

Abstract

Stochastic gradient descent (SGD) uses approximate gradients estimated from subsets of the training data and updates the parameters in an online fashion. This learning framework is attractive because it often requires much less training time in practice than batch training algorithms. However, L1-regularization, which is becoming popular in natural language processing because of its ability to produce compact models, cannot be efficiently applied in SGD training, due to the large dimensions of feature vectors and the fluctuations of approximate gradients. We present a simple method to solve these problems by penalizing the weights according to cumulative values for L1 penalty. We evaluate the effectiveness of our method in three applications: text chunking, named entity recognition, and part-of-speech tagging. Experimental results demonstrate that our method can produce compact and accurate models much more quickly than a state-of-the-art quasi-Newton method for L1-regularized log-linear models.

1 Introduction

Log-linear models (a.k.a maximum entropy models) are one of the most widely-used probabilistic models in the field of natural language processing (NLP). The applications range from simple classification tasks such as text classification and history-based tagging (Ratnaparkhi, 1996) to more complex structured prediction tasks such as part-of-speech (POS) tagging (Lafferty et al., 2001), syntactic parsing (Clark and Curran, 2004) and semantic role labeling (Toutanova et al., 2005). Log-linear models have a major advantage over other discriminative machine learning models such as support vector machines—their probabilistic output allows the information on the confidence of the decision to be used by other components in the text processing pipeline.

The training of log-liner models is typically performed based on the maximum likelihood criterion, which aims to obtain the weights of the features that maximize the conditional likelihood of the training data. In maximum likelihood training, *regularization* is normally needed to prevent the model from overfitting the training data,

The two most common regularization methods are called L1 and L2 regularization. L1 regularization penalizes the weight vector for its L1-norm (i.e. the sum of the absolute values of the weights), whereas L2 regularization uses its L2-norm. There is usually not a considerable difference between the two methods in terms of the accuracy of the resulting model (Gao et al., 2007), but L1 regularization has a significant advantage in practice. Because many of the weights of the features become zero as a result of L1-regularized training, the size of the model can be much smaller than that produced by L2-regularization. Compact models require less space on memory and storage, and enable the application to start up quickly. These merits can be of vital importance when the application is deployed in resource-tight environments such as cell-phones.

A common way to train a large-scale L1-regularized model is to use a quasi-Newton method. Kazama and Tsujii (2003) describe a method for training a L1-regularized log-linear model with a bound constrained version of the BFGS algorithm (Nocedal, 1980). Andrew and Gao (2007) present an algorithm called Orthant-Wise Limited-memory Quasi-Newton (OWL-QN), which can work on the BFGS algorithm without bound constraints and achieve faster convergence.

Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pages 477–485,
Suntec, Singapore, 2-7 August 2009.

An alternative approach to training a log-linear model is to use stochastic gradient descent (SGD) methods. SGD uses approximate gradients estimated from subsets of the training data and updates the weights of the features in an online fashion—the weights are updated much more frequently than batch training algorithms. This learning framework is attracting attention because it often requires much less training time in practice than batch training algorithms, especially when the training data is large and redundant. SGD was recently used for NLP tasks including machine translation (Tillmann and Zhang, 2006) and syntactic parsing (Smith and Eisner, 2008; Finkel et al., 2008). Also, SGD is very easy to implement because it does not need to use the Hessian information on the objective function. The implementation could be as simple as the perceptron algorithm.

Although SGD is a very attractive learning framework, the direct application of L1 regularization in this learning framework does not result in efficient training. The first problem is the inefficiency of applying the L1 penalty to the weights of all features. In NLP applications, the dimension of the feature space tends to be very large—it can easily become several millions, so the application of L1 penalty to all features significantly slows down the weight updating process. The second problem is that the naive application of L1 penalty in SGD does not always lead to compact models, because the approximate gradient used at each update is very noisy, so the weights of the features can be easily moved away from zero by those fluctuations.

In this paper, we present a simple method for solving these two problems in SGD learning. The main idea is to keep track of the total penalty and the penalty that has been applied to each weight, so that the L1 penalty is applied based on the difference between those cumulative values. That way, the application of L1 penalty is needed only for the features that are used in the current sample, and also the effect of noisy gradient is smoothed away.

We evaluate the effectiveness of our method by using linear-chain conditional random fields (CRFs) and three traditional NLP tasks, namely, text chunking (shallow parsing), named entity recognition, and POS tagging. We show that our enhanced SGD learning method can produce compact and accurate models much more quickly than the OWL-QN algorithm.

This paper is organized as follows. Section 2 provides a general description of log-linear models used in NLP. Section 3 describes our stochastic gradient descent method for L1-regularized log-linear models. Experimental results are presented in Section 4. Some related work is discussed in Section 5. Section 6 gives some concluding remarks.

2 Log-Linear Models

In this section, we briefly describe log-linear models used in NLP tasks and L1 regularization.

A log-linear model defines the following probabilistic distribution over possible structure $\mathbf{y}$ for input $\mathbf{x}$:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \sum_i w_i f_i(\mathbf{y}, \mathbf{x}),$$

where $f_i(\mathbf{y}, \mathbf{x})$ is a function indicating the occurrence of feature i, w_i is the weight of the feature, and $Z(\mathbf{x})$ is a partition (normalization) function:

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp \sum_i w_i f_i(\mathbf{y}, \mathbf{x}).$$

If the structure is a sequence, the model is called a linear-chain CRF model, and the marginal probabilities of the features and the partition function can be efficiently computed by using the forward-backward algorithm. The model is used for a variety of sequence labeling tasks such as POS tagging, chunking, and named entity recognition.

If the structure is a tree, the model is called a tree CRF model, and the marginal probabilities can be computed by using the inside-outside algorithm. The model can be used for tasks like syntactic parsing (Finkel et al., 2008) and semantic role labeling (Cohn and Blunsom, 2005).

2.1 Training

The weights of the features in a log-linear model are optimized in such a way that they maximize the regularized conditional log-likelihood of the training data:

$$\mathcal{L}_{\mathbf{w}} = \sum_{j=1}^{N} \log p(\mathbf{y}_j|\mathbf{x}_j; \mathbf{w}) - R(\mathbf{w}), \qquad (1)$$

where N is the number of training samples, $\mathbf{y}_j$ is the correct output for input $\mathbf{x}_j$, and $R(\mathbf{w})$ is the regularization term which prevents the model from overfitting the training data. In the case of L1 regularization, the term is defined as:

$$R(\mathbf{w}) = C \sum_i |w_i|,$$

where C is the meta-parameter that controls the degree of regularization, which is usually tuned by cross-validation or using the heldout data.

In what follows, we denote by $L(j, \mathbf{w})$ the conditional log-likelihood of each sample $\log p(\mathbf{y}_j|\mathbf{x}_j; \mathbf{w})$. Equation 1 is rewritten as:

$$\mathcal{L}_{\mathbf{w}} = \sum_{j=1}^{N} L(j, \mathbf{w}) - C \sum_i |w_i|. \qquad (2)$$

3 Stochastic Gradient Descent

SGD uses a small randomly-selected subset of the training samples to approximate the gradient of the objective function given by Equation 2. The number of training samples used for this approximation is called the *batch size*. When the batch size is N, the SGD training simply translates into gradient descent (hence is very slow to converge). By using a small batch size, one can update the parameters more frequently than gradient descent and speed up the convergence. The extreme case is a batch size of 1, and it gives the maximum frequency of updates and leads to a very simple perceptron-like algorithm, which we adopt in this work.[1]

Apart from using a single training sample to approximate the gradient, the optimization procedure is the same as simple gradient descent,[2] so the weights of the features are updated at training sample j as follows:

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \eta_k \frac{\partial}{\partial \mathbf{w}} (L(j, \mathbf{w}) - \frac{C}{N} \sum_i |w_i|),$$

where k is the iteration counter and η_k is the learning rate, which is normally designed to decrease as the iteration proceeds. The actual learning rate scheduling methods used in our experiments are described later in Section 3.3.

[1] In the actual implementation, we randomly shuffled the training samples at the beginning of each pass, and then picked them up sequentially.

[2] What we actually do here is gradient ascent, but we stick to the term "gradient descent".

3.1 L1 regularization

The update equation for the weight of each feature i is as follows:

$$w_i{}^{k+1} = w_i{}^k + \eta_k \frac{\partial}{\partial w_i} (L(j, \mathbf{w}) - \frac{C}{N} |w_i|).$$

The difficulty with L1 regularization is that the last term on the right-hand side of the above equation is not differentiable when the weight is zero. One straightforward solution to this problem is to consider a subgradient at zero and use the following update equation:

$$w_i{}^{k+1} = w_i{}^k + \eta_k \frac{\partial L(j, \mathbf{w})}{\partial w_i} - \frac{C}{N} \eta_k \text{sign}(w_i^k),$$

where $\text{sign}(x) = 1$ if $x > 0$, $\text{sign}(x) = -1$ if $x < 0$, and $\text{sign}(x) = 0$ if $x = 0$. In this paper, we call this weight updating method "SGD-L1 (Naive)".

This naive method has two serious problems. The first problem is that, at each update, we need to perform the application of L1 penalty to all features, including the features that are not used in the current training sample. Since the dimension of the feature space can be very large, it can significantly slow down the weight update process.

The second problem is that it does not produce a compact model, i.e. most of the weights of the features do not become zero as a result of training. Note that the weight of a feature does not become zero unless it happens to fall on zero exactly, which rarely happens in practice.

Carpenter (2008) describes an alternative approach. The weight updating process is divided into two steps. First, the weight is updated without considering the L1 penalty term. Then, the L1 penalty is applied to the weight to the extent that it does not change its sign. In other words, the weight is clipped when it crosses zero. Their weight update procedure is as follows:

$$w_i^{k+\frac{1}{2}} = w_i^k + \eta_k \left. \frac{\partial L(j, \mathbf{w})}{\partial w_i} \right|_{\mathbf{w}=\mathbf{w}^k},$$

if $w_i^{k+\frac{1}{2}} > 0$ **then**

$$w_i^{k+1} = \max(0, w_i^{k+\frac{1}{2}} - \frac{C}{N} \eta_k),$$

else if $w_i^{k+\frac{1}{2}} < 0$ **then**

$$w_i^{k+1} = \min(0, w_i^{k+\frac{1}{2}} + \frac{C}{N} \eta_k).$$

In this paper, we call this update method "SGD-L1 (Clipping)". It should be noted that this method

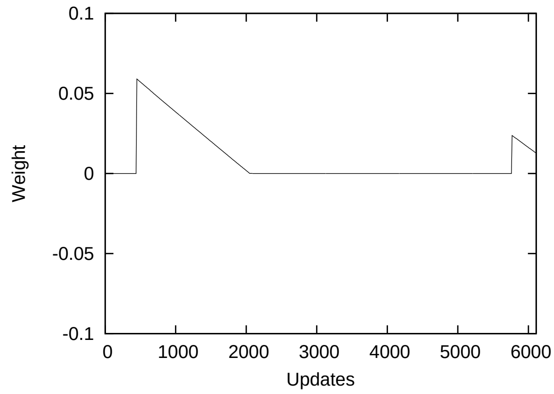

Figure 1: An example of weight updates.

is actually a special case of the FOLOS algorithm (Duchi and Singer, 2008) and the truncated gradient method (Langford et al., 2009).

The obvious advantage of using this method is that we can expect many of the weights of the features to become zero during training. Another merit is that it allows us to perform the application of L1 penalty in a lazy fashion, so that we do not need to update the weights of the features that are not used in the current sample, which leads to much faster training when the dimension of the feature space is large. See the aforementioned papers for the details. In this paper, we call this efficient implementation "SGD-L1 (Clipping + Lazy-Update)".

3.2 L1 regularization with cumulative penalty

Unfortunately, the clipping-at-zero approach does not solve all problems. Still, we often end up with many features whose weights are not zero. Recall that the gradient used in SGD is a crude approximation to the true gradient and is very noisy. The weight of a feature is, therefore, easily moved away from zero when the feature is used in the current sample.

Figure 1 gives an illustrative example in which the weight of a feature fails to become zero. The figure shows how the weight of a feature changes during training. The weight goes up sharply when it is used in the sample and then is pulled back toward zero gradually by the L1 penalty. Therefore, the weight fails to become zero if the feature is used toward the end of training, which is the case in this example. Note that the weight would become zero if the true (fluctuationless) gradient were used—at each update the weight would go up a little and be pulled back to zero straightaway.

Here, we present a different strategy for applying the L1 penalty to the weights of the features. The key idea is to smooth out the effect of fluctuating gradients by considering the cumulative effects from L1 penalty.

Let u_k be the absolute value of the total L1-penalty that each weight could have received up to the point. Since the absolute value of the L1 penalty does not depend on the weight and we are using the same regularization constant C for all weights, it is simply accumulated as:

$$u_k = \frac{C}{N} \sum_{t=1}^{k} \eta_t. \tag{3}$$

At each training sample, we update the weights of the features that are used in the sample as follows:

$$w_i^{k+\frac{1}{2}} = w_i^k + \eta_k \left. \frac{\partial L(j, \mathbf{w})}{\partial w_i} \right|_{\mathbf{w}=\mathbf{w}^k},$$

if $w_i^{k+\frac{1}{2}} > 0$ **then**

$$w_i^{k+1} = \max(0, w_i^{k+\frac{1}{2}} - (u_k + q_i^{k-1})),$$

else if $w_i^{k+\frac{1}{2}} < 0$ **then**

$$w_i^{k+1} = \min(0, w_i^{k+\frac{1}{2}} + (u_k - q_i^{k-1})),$$

where q_i^k is the total L1-penalty that w_i has actually received up to the point:

$$q_i^k = \sum_{t=1}^{k} (w_i^{t+1} - w_i^{t+\frac{1}{2}}). \tag{4}$$

This weight updating method penalizes the weight according to the difference between u_k and q_i^{k-1}. In effect, it forces the weight to receive the total L1 penalty that would have been applied if the weight had been updated by the true gradients, assuming that the current weight vector resides in the same orthant as the true weight vector.

It should be noted that this method is basically equivalent to a "SGD-L1 (Clipping + Lazy-Update)" method if we were able to use the true gradients instead of the stochastic gradients.

In this paper, we call this weight updating method "SGD-L1 (Cumulative)". The implementation of this method is very simple. Figure 2 shows the whole SGD training algorithm with this strategy in pseudo-code.

```
procedure TRAIN(C)
    u ← 0
    Initialize w_i and q_i with zero for all i
    for k = 0 to MaxIterations
        η ← LEARNINGRATE(k)
        u ← u + ηC/N
        Select sample j randomly
        UPDATEWEIGHTS(j)

procedure UPDATEWEIGHTS(j)
    for i ∈ features used in sample j
        w_i ← w_i + η ∂L(j,w)/∂w_i
        APPLYPENALTY(i)

procedure APPLYPENALTY(i)
    z ← w_i
    if w_i > 0 then
        w_i ← max(0, w_i − (u + q_i))
    else if w_i < 0 then
        w_i ← min(0, w_i + (u − q_i))
    q_i ← q_i + (w_i − z)
```

Figure 2: Stochastic gradient descent training with cumulative L1 penalty. z is a temporary variable.

3.3 Learning Rate

The scheduling of learning rates often has a major impact on the convergence speed in SGD training.

A typical choice of learning rate scheduling can be found in (Collins et al., 2008):

$$\eta_k = \frac{\eta_0}{1 + k/N}, \tag{5}$$

where η_0 is a constant. Although this scheduling guarantees ultimate convergence, the actual speed of convergence can be poor in practice (Darken and Moody, 1990).

In this work, we also tested simple exponential decay:

$$\eta_k = \eta_0 \alpha^{-k/N}, \tag{6}$$

where α is a constant. In our experiments, we found this scheduling more practical than that given in Equation 5. This is mainly because exponential decay sweeps the range of learning rates more smoothly—the learning rate given in Equation 5 drops too fast at the beginning and too slowly at the end.

It should be noted that exponential decay is not a good choice from a theoretical point of view, because it does not satisfy one of the necessary conditions for convergence—the sum of the learning rates must diverge to infinity (Spall, 2005). However, this is probably not a big issue for practitioners because normally the training has to be terminated at a certain number of iterations in practice.[3]

4 Experiments

We evaluate the effectiveness our training algorithm using linear-chain CRF models and three NLP tasks: text chunking, named entity recognition, and POS tagging.

To compare our algorithm with the state-of-the-art, we present the performance of the OWL-QN algorithm on the same data. We used the publicly available OWL-QN optimizer developed by Andrew and Gao.[4] The meta-parameters for learning were left unchanged from the default settings of the software: the convergence tolerance was 1e-4; and the L-BFGS memory parameter was 10.

4.1 Text Chunking

The first set of experiments used the text chunking data set provided for the CoNLL 2000 shared task.[5] The training data consists of 8,936 sentences in which each token is annotated with the "IOB" tags representing text chunks such as noun and verb phrases. We separated 1,000 sentences from the training data and used them as the heldout data. The test data provided by the shared task was used only for the final accuracy report.

The features used in this experiment were unigrams and bigrams of neighboring words, and unigrams, bigrams and trigrams of neighboring POS tags.

To avoid giving any advantage to our SGD algorithms over the OWL-QN algorithm in terms of the accuracy of the resulting model, the OWL-QN algorithm was used when tuning the regularization parameter C. The tuning was performed in such a way that it maximized the likelihood of the heldout data. The learning rate parameters for SGD were then tuned in such a way that they maximized the value of the objective function in 30 passes. We first determined η_0 by testing 1.0, 0.5, 0.2, and 0.1. We then determined α by testing 0.9, 0.85, and 0.8 with the fixed η_0.

[3] This issue could also be sidestepped by, for example, adding a small $O(1/k)$ term to the learning rate.

[4] Available from the original developers' websites: http://research.microsoft.com/en-us/people/galena/ or http://research.microsoft.com/en-us/um/people/jfgao/

[5] http://www.cnts.ua.ac.be/conll2000/chunking/

	Passes	$\mathcal{L}_{\mathbf{w}}/N$	# Features	Time (sec)	F-score
OWL-QN	160	-1.583	18,109	598	93.62
SGD-L1 (Naive)	30	-1.671	455,651	1,117	93.64
SGD-L1 (Clipping + Lazy-Update)	30	-1.671	87,792	144	93.65
SGD-L1 (Cumulative)	30	-1.653	28,189	149	93.68
SGD-L1 (Cumulative + Exponential-Decay)	30	-1.622	23,584	148	93.66

Table 1: CoNLL-2000 Chunking task. Training time and accuracy of the trained model on the test data.

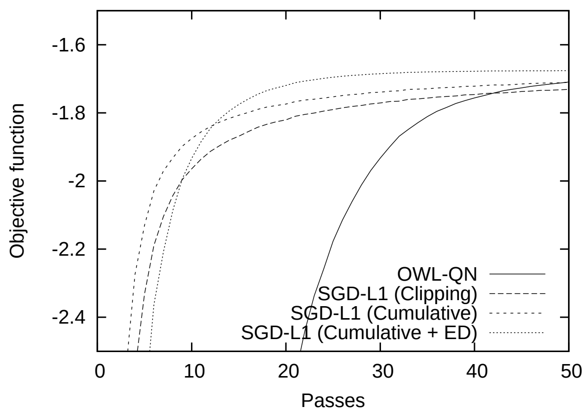

Figure 3: CoNLL 2000 chunking task: Objective

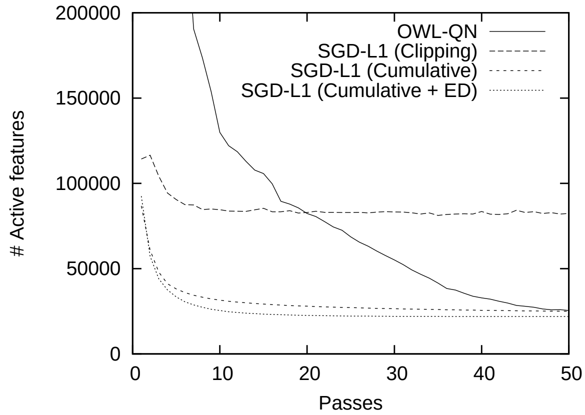

Figure 4: CoNLL 2000 chunking task: Number of active features.

Figures 3 and 4 show the training process of the model. Each figure contains four curves representing the results of the OWL-QN algorithm and three SGD-based algorithms. "SGD-L1 (Cumulative + ED)" represents the results of our cumulative penalty-based method that uses exponential decay (ED) for learning rate scheduling.

Figure 3 shows how the value of the objective function changed as the training proceeded. SGD-based algorithms show much faster convergence than the OWL-QN algorithm. Notice also that "SGD-L1 (Cumulative)" improves the objective slightly faster than "SGD-L1 (Clipping)". The result of "SGD-L1 (Naive)" is not shown in this figure, but the curve was almost identical to that of "SGD-L1 (Clipping)".

Figure 4 shows the numbers of active features (the features whose weight are not zero). It is clearly seen that the clipping-at-zero approach fails to reduce the number of active features, while our algorithms succeeded in reducing the number of active features to the same level as OWL-QN.

We then trained the models using the whole training data (including the heldout data) and evaluated the accuracy of the chunker on the test data. The number of passes performed over the training data in SGD was set to 30. The results are shown in Table 1. The second column shows the number of passes performed in the training. The third column shows the final value of the objective function per sample. The fourth column shows the number of resulting active features. The fifth column show the training time. The last column shows the f-score (harmonic mean of recall and precision) of the chunking results. There was no significant difference between the models in terms of accuracy. The naive SGD training took much longer than OWL-QN because of the overhead of applying L1 penalty to all dimensions.

Our SGD algorithms finished training in 150 seconds on Xeon 2.13GHz processors. The CRF++ version 0.50, a popular CRF library developed by Taku Kudo,[6] is reported to take 4,021 seconds on Xeon 3.0GHz processors to train the model using a richer feature set.[7] CRFsuite version 0.4, a much faster library for CRFs, is reported to take 382 seconds on Xeon 3.0GHz, using the same feature set as ours.[8] Their library uses the OWL-QN algorithm for optimization. Although direct comparison of training times is not impor-

[6] http://crfpp.sourceforge.net/

[7] http://www.chokkan.org/software/crfsuite/benchmark.html

[8] ditto

tant due to the differences in implementation and hardware platforms, these results demonstrate that our algorithm can actually result in a very fast implementation of a CRF trainer.

4.2 Named Entity Recognition

The second set of experiments used the named entity recognition data set provided for the BioNLP/NLPBA 2004 shared task (Kim et al., 2004).[9] The training data consist of 18,546 sentences in which each token is annotated with the "IOB" tags representing biomedical named entities such as the names of proteins and RNAs.

The training and test data were preprocessed by the GENIA tagger,[10] which provided POS tags and chunk tags. We did not use any information on the named entity tags output by the GENIA tagger. For the features, we used unigrams of neighboring chunk tags, substrings (shorter than 10 characters) of the current word, and the shape of the word (e.g. "IL-2" is converted into "AA-#"), on top of the features used in the text chunking experiments.

The results are shown in Figure 5 and Table 2. The trend in the results is the same as that of the text chunking task: our SGD algorithms show much faster convergence than the OWL-QN algorithm and produce compact models.

Okanohara et al. (2006) report an f-score of 71.48 on the same data, using semi-Markov CRFs.

4.3 Part-Of-Speech Tagging

The third set of experiments used the POS tagging data in the Penn Treebank (Marcus et al., 1994). Following (Collins, 2002), we used sections 0-18 of the Wall Street Journal (WSJ) corpus for training, sections 19-21 for development, and sections 22-24 for final evaluation. The POS tags were extracted from the parse trees in the corpus. All experiments for this work, including the tuning of features and parameters for regularization, were carried out using the training and development sets. The test set was used only for the final accuracy report.

It should be noted that training a CRF-based POS tagger using the whole WSJ corpus is not a trivial task and was once even deemed impractical in previous studies. For example, Wellner and Vilain (2006) abandoned maximum likelihood training because it was "prohibitive" (7-8 days for sections 0-18 of the WSJ corpus).

For the features, we used unigrams and bigrams of neighboring words, prefixes and suffixes of the current word, and some characteristics of the word. We also normalized the current word by lowering capital letters and converting all the numerals into '#', and used the normalized word as a feature.

The results are shown in Figure 6 and Table 3. Again, the trend is the same. Our algorithms finished training in about 30 minutes, producing accurate models that are as compact as that produced by OWL-QN.

Shen et al., (2007) report an accuracy of 97.33% on the same data set using a perceptron-based bidirectional tagging model.

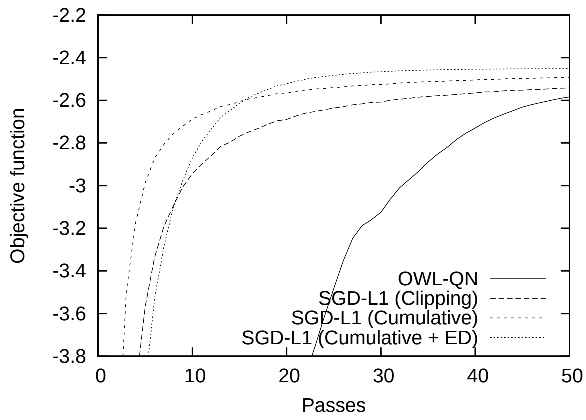

Figure 5: NLPBA 2004 named entity recognition task: Objective.

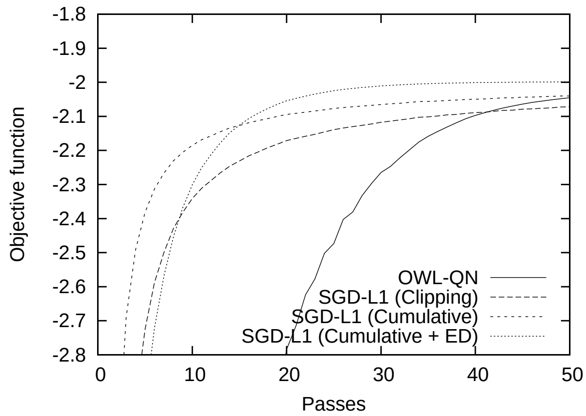

Figure 6: POS tagging task: Objective.

5 Discussion

An alternative approach to producing compact models for log-linear models is to reformulate the

[9] The data is available for download at http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/ERtask/report.html

[10] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/tagger/

	Passes	$\mathcal{L}_{\mathbf{w}}/N$	# Features	Time (sec)	F-score
OWL-QN	161	-2.448	30,710	2,253	71.76
SGD-L1 (Naive)	30	-2.537	1,032,962	4,528	71.20
SGD-L1 (Clipping + Lazy-Update)	30	-2.538	279,886	585	71.20
SGD-L1 (Cumulative)	30	-2.479	31,986	631	71.40
SGD-L1 (Cumulative + Exponential-Decay)	30	-2.443	25,965	631	71.63

Table 2: NLPBA 2004 Named entity recognition task. Training time and accuracy of the trained model on the test data.

	Passes	$\mathcal{L}_{\mathbf{w}}/N$	# Features	Time (sec)	Accuracy
OWL-QN	124	-1.941	50,870	5,623	97.16%
SGD-L1 (Naive)	30	-2.013	2,142,130	18,471	97.18%
SGD-L1 (Clipping + Lazy-Update)	30	-2.013	323,199	1,680	97.18%
SGD-L1 (Cumulative)	30	-1.987	62,043	1,777	97.19%
SGD-L1 (Cumulative + Exponential-Decay)	30	-1.954	51,857	1,774	97.17%

Table 3: POS tagging on the WSJ corpus. Training time and accuracy of the trained model on the test data.

problem as a L1-constrained problem (Lee et al., 2006), where the conditional log-likelihood of the training data is maximized under a fixed constraint of the L1-norm of the weight vector. Duchi et al. (2008) describe efficient algorithms for projecting a weight vector onto the L1-ball. Although L1-regularized and L1-constrained learning algorithms are not directly comparable because the objective functions are different, it would be interesting to compare the two approaches in terms of practicality. It should be noted, however, that the efficient algorithm presented in (Duchi et al., 2008) needs to employ a red-black tree and is rather complex.

In SGD learning, the need for tuning the meta-parameters for learning rate scheduling can be annoying. In the case of exponential decay, the setting of $\alpha = 0.85$ turned out to be a good rule of thumb in our experiments—it always produced near best results in 30 passes, but the other parameter η_0 needed to be tuned. It would be very useful if those meta-parameters could be tuned in a fully automatic way.

There are some sophisticated algorithms for adaptive learning rate scheduling in SGD learning (Vishwanathan et al., 2006; Huang et al., 2007). However, those algorithms use second-order information (i.e. Hessian information) and thus need access to the weights of the features that are not used in the current sample, which should slow down the weight updating process for the same reason discussed earlier. It would be interesting to investigate whether those sophisticated learning scheduling algorithms can actually result in fast training in large-scale NLP tasks.

6 Conclusion

We have presented a new variant of SGD that can efficiently train L1-regularized log-linear models. The algorithm is simple and extremely easy to implement.

We have conducted experiments using CRFs and three NLP tasks, and demonstrated empirically that our training algorithm can produce compact and accurate models much more quickly than a state-of-the-art quasi-Newton method for L1-regularization.

Acknowledgments

We thank N. Okazaki, N. Yoshinaga, D. Okanohara and the anonymous reviewers for their useful comments and suggestions. The work described in this paper has been funded by the Biotechnology and Biological Sciences Research Council (BBSRC; BB/E004431/1). The research team is hosted by the JISC/BBSRC/EPSRC sponsored National Centre for Text Mining.

References

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L1-regularized log-linear models. In *Proceedings of ICML*, pages 33–40.

Bob Carpenter. 2008. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. Technical report, Alias-i.

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of COLING 2004*, pages 103–110.

Trevor Cohn and Philip Blunsom. 2005. Semantic role labeling with tree conditional random fields. In *Proceedings of CoNLL*, pages 169–172.

Michael Collins, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L. Bartlett. 2008. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *The Journal of Machine Learning Research (JMLR)*, 9:1775–1822.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8.

Christian Darken and John Moody. 1990. Note on learning rate schedules for stochastic optimization. In *Proceedings of NIPS*, pages 832–838.

Juhn Duchi and Yoram Singer. 2008. Online and batch learning using forward-looking subgradients. In *NIPS Workshop: OPT 2008 Optimization for Machine Learning*.

Juhn Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of ICML*, pages 272–279.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08:HLT*, pages 959–967.

Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. 2007. A comparative study of parameter estimation methods for statistical natural language processing. In *Proceedings of ACL*, pages 824–831.

Han-Shen Huang, Yu-Ming Chang, and Chun-Nan Hsu. 2007. Training conditional random fields by periodic step size adaptation for large-scale text mining. In *Proceedings of ICDM*, pages 511–516.

Jun'ichi Kazama and Jun'ichi Tsujii. 2003. Evaluation and extension of maximum entropy models with inequality constraints. In *Proceedings of EMNLP 2003*.

J.-D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, and N. Collier. 2004. Introduction to the bio-entity recognition task at JNLPBA. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 70–75.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, pages 282–289.

John Langford, Lihong Li, and Tong Zhang. 2009. Sparse online learning via truncated gradient. *The Journal of Machine Learning Research (JMLR)*, 10:777–801.

Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng. 2006. Efficient l1 regularized logistic regression. In *Proceedings of AAAI-06*, pages 401–408.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Jorge Nocedal. 1980. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782.

Daisuke Okanohara, Yusuke Miyao, Yoshimasa Tsuruoka, and Jun'ichi Tsujii. 2006. Improving the scalability of semi-markov conditional random fields for named entity recognition. In *Proceedings of COLING/ACL*, pages 465–472.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP 1996*, pages 133–142.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of ACL*, pages 760–767.

David Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of EMNLP*, pages 145–156.

James C. Spall. 2005. *Introduction to Stochastic Search and Optimization*. Wiley-IEEE.

Christoph Tillmann and Tong Zhang. 2006. A discriminative global training algorithm for statistical MT. In *Proceedings of COLING/ACL*, pages 721–728.

Kristina Toutanova, Aria Haghighi, and Christopher Manning. 2005. Joint learning improves semantic role labeling. In *Proceedings of ACL*, pages 589–596.

S. V. N. Vishwanathan, Nicol N. Schraudolph, Mark W. Schmidt, and Kevin P. Murphy. 2006. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of ICML*, pages 969–976.

Ben Wellner and Marc Vilain. 2006. Leveraging machine readable dictionaries in discriminative sequence models. In *Proceedings of LREC 2006*.