

Fast Newton-CG Method for Batch Learning of Conditional Random Fields

Yuta Tsuboi
IBM Research - Tokyo
Kanagawa, Japan.
yutat@jp.ibm.com

Yuya Unno *
Preferred Infrastructure, Inc.
Tokyo, Japan.
unno@preferred.jp

Hisashi Kashima
the University of Tokyo
Tokyo, Japan.
kashima@mist.i.u-tokyo.ac.jp

Naoaki Okazaki †
Tohoku University
Sendai, Japan.
okazaki@ecei.tohoku.ac.jp

Abstract

We propose a fast batch learning method for linear-chain Conditional Random Fields (CRFs) based on Newton-CG methods. Newton-CG methods are a variant of Newton method for high-dimensional problems. They only require the Hessian-vector products instead of the full Hessian matrices.

To speed up Newton-CG methods for the CRF learning, we derive a novel dynamic programming procedure for the Hessian-vector products of the CRF objective function. The proposed procedure can reuse the byproducts of the time-consuming gradient computation for the Hessian-vector products to drastically reduce the total computation time of the Newton-CG methods.

In experiments with tasks in natural language processing, the proposed method outperforms a conventional quasi-Newton method. Remarkably, the proposed method is competitive with online learning algorithms that are fast but unstable.

Introduction

Linear-chain *Conditional Random Fields* (CRFs) model the conditional probability of output sequences (Lafferty, McCallum, and Pereira 2001). They are simple but have been applied to a variety of sequential labeling problems, including natural language processing (Sha and Pereira 2003) and bioinformatics (Chen, Chen, and Brent 2008). The learning task of CRFs can be regarded as the unconstrained minimization of the regularized negative log-likelihood function. Since training CRF models can be computationally intensive, we want an optimization method that converges rapidly.

In unconstrained optimization, we minimize an objective function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that depends on the d dimensional parameter vector $\theta \in \mathbb{R}^d$, without constraints on the values of θ . Optimization algorithms generate a sequence of iterates $\{\theta_k\}_{k=0}^{\infty}$. In each iteration, typical algorithms try to move from the current point θ_k to a new point θ_{k+1} along a search direction s_k such that $f_{k+1} < f_k$ where $f_k \equiv f(\theta_k)$. The gradient descent direction $-g_k \equiv -\nabla_{\theta} f_k$

is the most obvious choice for the search direction. Another choice is the *Newton step* $-(H_k)^{-1}g_k \in \mathbb{R}^{d \times d}$, where $H_k \equiv \nabla_{\theta}^2 f_k$ is the Hessian matrix of f . The Newton step is the minimizer of the second order Taylor approximation of f : $f(\theta_k + s) \approx f_k + g_k^{\top} s + \frac{1}{2} s^{\top} H_k s$. Optimization methods using second order information have a fast rate of local convergence.

Since the explicit computation of the $d \times d$ Hessian matrix is not practical for large d , *Newton-CG* methods were developed for large problems (Nocedal and Wright 2006). The Newton step is the solution of the *Newton equation*: $H_k s = -g_k$. Newton-CG methods use a *Conjugate Gradient* (CG) method to solve the Newton equation. Since the CG method only requires the Hessian-vector products of the form $H_k r \in \mathbb{R}^d$ for an arbitrary vector $r \in \mathbb{R}^d$, Newton-CG methods are less resource demanding. In addition, Newton-CG methods efficiently find a sufficiently good search direction using the adaptive control of the quality of the Newton step.

Here are the contributions of this paper. First, to the best of our knowledge, this is the first application of Newton-CG methods to CRF learning. Second, we have devised a procedure that computes the Hessian-vector products of the CRF objective function in polynomial time. We show that the additional computational costs for the Hessian-vector products are relatively small compared to the computational costs for the gradient computations. Finally, we show that the computations for the repetitive Hessian-vector products in the CG iterations can be reduced by reusing the marginal probabilities, which are the byproducts of the costly gradient computation. Since the total effort for Newton-CG methods is dominated by the cumulative costs of the CG iterations, the proposed method can significantly accelerate Newton-CG methods for CRF learning.

In experiments on natural language tasks, the proposed method outperforms a conventional quasi-Newton method. In addition, the experimental results also show the proposed method is competitive with online learning algorithms whose generalization/optimization performance is task-dependent. Online learning algorithms have been attracting attention because of their capabilities to handle huge amounts of data with acceptable accuracy (Bottou and Bousquet 2008). However, they have considerable disadvantages compared to batch learning, such as the lack of stopping cri-

*This work was done when YU was at IBM Research - Tokyo.

†This work was done when NO was at the University of Tokyo.
Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

teria and the tuning efforts for learning-rates. Also, batch algorithms show more stable and task-independent performance. Therefore, the proposed fast method will broaden the application field for batch CRF learning.

Conditional Random Fields

We briefly review the linear-chain CRF models and their parameter estimation problems (Lafferty, McCallum, and Pereira 2001). Let $\mathbf{x} = (x_1, \dots, x_T) \in \mathbf{X}$ denote an observed sequence of $x \in X$, $\mathbf{y} = (y_1, \dots, y_T) \in \mathbf{Y}$ denote a label sequence of $y \in Y$, $\Phi(\mathbf{x}, \mathbf{y}) : \mathbf{X} \times \mathbf{Y} \rightarrow \mathbb{R}^d$ denote a map from a pair of \mathbf{x} and \mathbf{y} to an arbitrary feature vector of d dimensions, and $\theta \in \mathbb{R}^d$ denote the model parameters. CRFs model the conditional probability of \mathbf{y} given \mathbf{x} as $P_\theta(\mathbf{y}|\mathbf{x}) = \exp(\theta^\top \Phi(\mathbf{x}, \mathbf{y})) / Z$ where the denominator is the normalization term: $Z = \sum_{\mathbf{y} \in \mathbf{Y}} \exp(\theta^\top \Phi(\mathbf{x}, \mathbf{y}))$. Once θ has been estimated, the label sequence can be predicted as $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathbf{Y}} P_\theta(\mathbf{y}|\mathbf{x})$. In the sequel, the argument θ of functions is omitted unless required for clarity.

Using a training set $D \equiv \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$, the optimal parameter is obtained as the minimizer of the regularized negative log likelihood function:

$$f(\theta) = - \sum_{(\mathbf{x}, \mathbf{y}) \in D} [\theta^\top \Phi(\mathbf{x}, \mathbf{y}) - \ln Z] + \frac{\|\theta\|^2}{2\sigma^2},$$

where the final term is a Gaussian prior with mean 0 and variance σ^2 . This CRF learning is stated as an unconstrained minimization problem for the convex non-linear function f . The typical optimizers for CRFs use the gradient of f :

$$\mathbf{g} = - \sum_{(\mathbf{x}, \mathbf{y}) \in D} \mathbf{g}(\mathbf{x}, \mathbf{y}) + \frac{\theta}{\sigma^2},$$

where the instance-wise gradient is evaluated as

$$\mathbf{g}(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}, \mathbf{y}) - \mathbb{E}_{P_\theta}(\Phi(\mathbf{x}, \mathbf{y}))$$

with $\mathbb{E}_{P_\theta}(\Phi(\mathbf{x}, \mathbf{y})) = \sum_{\mathbf{y} \in \mathbf{Y}} P_\theta(\mathbf{y}|\mathbf{x}) \Phi(\mathbf{x}, \mathbf{y})$.

Although $\mathbb{E}_{P_\theta}(\Phi(\mathbf{x}, \mathbf{y}))$ includes the sum of \mathbf{Y} of exponential size, a polynomial time algorithm, which is known as the *forward-backward* algorithm, can be used for the linear-chain CRFs. Without going into detail, the key ingredient is the marginal probabilities of a label pair (y_{t-1}, y_t) at position t which can be evaluated as

$$P_\theta(y_{t-1}=i, y_t=j|\mathbf{x}) = \exp(\alpha[t-1, i] + \theta^\top \phi(\mathbf{x}, i, j) + \beta[t, j] - \ln Z), \quad (1)$$

where we assume the feature function can be decomposed as $\Phi(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T+1} \phi(\mathbf{x}, y_{t-1}, y_t)$. Then we can evaluate $\mathbb{E}_{P_\theta}(\Phi(\mathbf{x}, \mathbf{y}))$ in $O(T|Y|^2)$ time when $\alpha[t, j] = \ln \sum_{i \in Y} \exp(\alpha[t-1, i] + \theta^\top \phi(\mathbf{x}, i, j))$ and $\beta[t, i] = \ln \sum_{j \in Y} \exp(\theta^\top \phi(\mathbf{x}, i, j) + \beta[t+1, j])$ have been pre-computed. Note that the marginal probabilities of a single pair $P_\theta(y_t|\mathbf{x})$ can also be evaluated in the same manner.

Newton-CG Methods

The essential idea of the Newton-CG methods is that we do not need to compute Hessian exactly, but we only need a good enough search direction.

The procedure of Newton-CG methods have two nested layers of iterations: an inner conjugate gradient (CG) procedure finds the Newton step, and an outer procedure modifies the Newton step to ensure global convergence. In the sequel, outer and inner iterates are indexed by k and ℓ , respectively.

At each outer iteration, the CG procedure finds the approximation of the Newton step by solving the Newton equation:

$$\mathbf{H}\mathbf{s} = -\mathbf{g}, \quad (2)$$

where we denote $\mathbf{H} \equiv \mathbf{H}_k$ and $\mathbf{g} \equiv \mathbf{g}_k$ since the value of θ_k does not change in the inner iterations. The CG method is an iterative solver for linear systems, and generates a set of *conjugate* directions $\{\mathbf{r}_\ell\}_{\ell=1}^d$, where $\mathbf{r}_\ell^\top \mathbf{H} \mathbf{r}_h = 0$ for all $\ell \neq h$. In fact, we can solve Eq. (2) in at most d steps of the CG procedure. Since the total effort for Newton-CG methods is dominated by the cumulative sum of the CG iterations performed, it is preferable to computing approximations to the Newton step. The accuracy measure of the Newton step is based on the norm of the residual $p = \|\mathbf{H} \mathbf{r}_\ell + \mathbf{g}\|$ of Eq. (2). Typically, we terminate the CG iterations when $p \leq \xi_k \|\mathbf{g}_k\|$, where the sequence $\{\xi_k | 0 < \xi_k < 1\}$.

Since the objective function of CRFs is not a quadratic form, the function value may not be guaranteed to decrease in each Newton step. For non-linear problems, the *line search* strategy or *trust-region* strategy is commonly employed in the outer iterations. In both strategies, if $\lim_{k \rightarrow \infty} \xi_k = 0$, we have superlinear convergence (Nocedal and Wright 2006). In our experiments, we used the trust-region strategy. In addition, the stopping condition of the outer loop is typically the magnitude of $\|\mathbf{g}_k\|$.

The advantage of Newton-CG methods is that they adaptively control the accuracy of the solution without loss of the rapid convergence properties of the Newton method. In addition, the CG procedure only requires the Hessian-vector products $\mathbf{H}\mathbf{r}$ for an arbitrary vector \mathbf{r} . In the next section, we describe an efficient algorithm for the $\mathbf{H}\mathbf{r}$ computations of the CRF objective function.

Hessian-vector Products of CRFs

In this section, we propose a novel dynamic programming (DP) procedure which can calculate the Hessian-vector product in $O(T|Y|^{\kappa+1})$ time and space for the κ -th order CRFs. This procedure computes the Hessian-vector products using the marginal probabilities computed in the performance-critical point of the CRF gradient computations. Since, in the inner CG loop, the Hessian with a fixed θ is iteratively multiplied by different kinds of vectors, the proposed method accelerates the Newton-CG methods by reusing the same marginal probabilities.

Let \mathbf{r} be an arbitrary vector, then the product of the Hessian \mathbf{H} and \mathbf{r} is stated as

$$\mathbf{H}\mathbf{r} = \sum_{(\mathbf{x}, \mathbf{y}) \in D} \mathbf{H}(\mathbf{x}, \mathbf{y})\mathbf{r} + \frac{\mathbf{r}}{\sigma^2},$$

where the instance-wise Hessian-vector product is defined as

$$\mathbf{H}(\mathbf{x}, \mathbf{y})\mathbf{r} = \sum_{\mathbf{y} \in \mathbf{Y}} P_\theta(\mathbf{y}|\mathbf{x}) \Phi(\mathbf{x}, \mathbf{y}) \mathbf{g}(\mathbf{x}, \mathbf{y})^\top \mathbf{r}. \quad (3)$$

Since Eq. (3) also includes the sum of all of the possible configurations Y of exponential size, we propose a DP procedure. In addition, instead of $\Phi(\mathbf{x}, \mathbf{y})\mathbf{g}(\mathbf{x}, \mathbf{y})^\top \in \mathbb{R}^{d \times d}$, $\mathbf{g}(\mathbf{x}, \mathbf{y})^\top \mathbf{r} \in \mathbb{R}$ is evaluated at first to avoid materializing matrixes in the procedure.

Eq. (3) can be restate as a point-wise summation:

$$\mathbf{H}(\mathbf{x}, \mathbf{y})\mathbf{r} = \sum_{t=1}^T \sum_{i,j \in Y} \phi(\mathbf{x}, i, j)M(t, i, j)$$

with a scalar valued function

$$M(t, i, j) = \sum_{\tilde{\mathbf{y}}: \tilde{y}_{t-1}=i \wedge \tilde{y}_t=j} P(\tilde{\mathbf{y}}|\mathbf{x})\mathbf{g}(\mathbf{x}, \tilde{\mathbf{y}})^\top \mathbf{r},$$

where $\sum_{\tilde{\mathbf{y}}: \tilde{y}_{t-1}=i \wedge \tilde{y}_t=j}$ indicates summation over all of the label sequences with the $(t-1)$ -st and t -th labels fixed as i and j , respectively. Without loss of generality, we assume the first order Markov model. Under this assumption, the conditional probability $P(\mathbf{y}|\mathbf{x})$ can also be decomposed into

$$P(\mathbf{y}|\mathbf{x}) = P(y_1|y_2, \mathbf{x}) \cdots P(y_{t-2}|y_{t-1}, \mathbf{x}) \\ P(y_{t-1}, y_t|\mathbf{x})P(y_{t+1}|y_t, \mathbf{x}) \cdots P(y_T|y_{T-1}, \mathbf{x}).$$

Note that $P(S)P(S|y_1, \mathbf{x}) = 1$ and $P(E)P(E|y_T, \mathbf{x}) = 1$ are omitted where S and E are special label variables to encode the start and end of a sequence, respectively. Then we can restate $M(t, i, j)$ in a recursive form as

$$M(t, i, j) = P_\theta(y_{t-1}=i, y_t=j|\mathbf{x}) \\ [A[t-1, i] + \phi(\mathbf{x}, i, j)^\top \mathbf{r} + B[t, j] - \mathbb{E}_{P_\theta}(\Phi(\mathbf{x}, \mathbf{y}))^\top \mathbf{r}].$$

The tables $A[t, j]$ and $B[t, i]$ are defined as

$$A[t, j] = \begin{cases} 0 & \text{if } t = 0 \\ \phi(\mathbf{x}, S, j)^\top \mathbf{r} & \text{else if } t=1 \\ \sum_{i \in Y} P_\theta(y_{t-1}=i|y_t=j, \mathbf{x}) \\ (\phi(\mathbf{x}, i, j)^\top \mathbf{r} + A[t-1, i]) & \text{otherwise, and} \end{cases}$$

$$B[t, i] = \begin{cases} \phi(\mathbf{x}, i, E)^\top \mathbf{r} & \text{if } t=T \\ \sum_{j \in Y} P_\theta(y_{t+1}=j|y_t=i, \mathbf{x}) \\ (\phi(\mathbf{x}, i, j)^\top \mathbf{r} + B[t+1, j]) & \text{otherwise.} \end{cases}$$

When we pre-compute the entries of A for $\{t|0 \leq t \leq T\}$ and B for $\{t|1 \leq t \leq T+1\}$, the evaluation of Eq. (3) requires $O(T|Y|^2)$ time. In general, the Hessian-vector products require $(O(T|Y|^{\kappa+1}))$ time and space for the κ -th order Markov model.

The important property is that we use the marginal probabilities Eq. (1) in the proposed procedure. The conditional probabilities in the definition of A and B can be obtained by the division of the marginal probabilities: $P_\theta(y_{t-1}|y_t, \mathbf{x}) = P_\theta(y_{t-1}, y_t|\mathbf{x})/P_\theta(y_t|\mathbf{x})$ and $P_\theta(y_{t+1}|y_t, \mathbf{x}) = P_\theta(y_t, y_{t+1}|\mathbf{x})/P_\theta(y_t|\mathbf{x})$. Note that the proposed procedure is composed of only simple arithmetic except for the computations of the marginal probabilities. The performance-critical point of the CRF gradient computation is $O(T|Y|^{\kappa+1})$ exponential operations in Eq. (1), since they are more expensive computations than simple

arithmetic operations (Chen, Chen, and Brent 2008). For example, an exponential operation is 15 times more expensive than simple addition on our experimental platform. Thus, the additional costs of the Hessian-vector product computations are relatively smaller than the cost of the gradient computations.

This property is highly effective for the repetitive computations of the Hessian-vector product $\mathbf{H}_k \mathbf{r}_\ell$ in the inner CG loop of the Newton-CG methods. For all of the different \mathbf{r}_ℓ , we can share the same marginal probabilities calculated in the evaluation of \mathbf{g}_k . The reuse of the marginal probabilities can significantly reduce the computation time of the CG loop compared to the repetition of the marginal probability computations. Note that, since the storage for the marginal probabilities requires $\Omega(NT|Y|^{\kappa+1})$ space where N is the number of instances, it may not fit in physical memory for a large data set. In such cases, we can store the marginal probabilities for $c < N$ instances, which is a subset of D , and compute the marginal probabilities for the remaining $(N-c)$ instances in the CG loop.

Related Work

The most popular offline algorithm for CRF learning is the *limited-memory BFGS* (LBFGS) which is a quasi-Newton method that approximates the inverse of the Hessian \mathbf{H}^{-1} from the m most recent changes of the parameters and the gradients where $m \ll d$ (Sha and Pereira 2003). Thus, it stores only $2m$ vectors of length d that implicitly represent the approximations. However, Lin et al. (2008) show that a Newton-CG method outperforms the LBFGS method for large learning tasks of binary *logistic regression*. To the best of our knowledge, this is the first application of Newton-CG methods to the training of CRFs, which are the generalizations of logistic regression.

Recently, online learning algorithms have received great attention in the machine learning communities. One of the popular online learning algorithms is *Stochastic Gradient Descent* (SGD) which is applicable to the learning task of any differentiable loss function including the CRF loss function. For each iteration k , SGD for CRFs uses a gradient evaluated as $\mathbf{g}_k = -\mathbf{g}(\mathbf{x}, \mathbf{y}) + \theta/(N\sigma^2)$, where $(\mathbf{x}, \mathbf{y}) \in D$ is a random sample. For each instance, the SGD updates the parameter θ with this stochastic gradient: $\theta_{k+1} = \theta_k - \eta_k \mathbf{g}_k$, where $\eta_k > 0$ is the learning rate parameter. Although the optimization accuracy of SGD is lower than that of batch algorithms, it may find acceptable solutions in terms of testing performance. Major drawbacks of SGD are the tuning effort on the learning-rate parameter and the lack of the stopping criteria (Spall 2003). Meanwhile, for batch algorithms, there are the established problem-independent values for the tuning parameters and the stopping conditions.

Collins et al. (2008) proposed an interesting *Exponentiated Gradient* (EG) algorithm for log-linear models including CRFs. Although, the EG algorithm is similar to SGD in that both process one instance at a time, the EG corresponds to block-coordinate ascent using the dual formulation of the CRF objective function, and therefore uses a deterministic gradient with respect to the coordinate being updated. Since

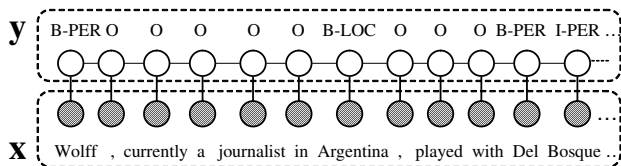


Figure 1: The named entity labels for an example sentence: Words labeled with O are non-named entities. The B-XXX label is used for the first word in a named entity of type XXX and I-XXX is used for all other continued words in named entities of type XXX. PER represents persons and LOC represents locations.

the deterministic value of the dual function is also available, the learning-rate of the EG can be adjusted based on the improvement of the dual. Note however, that there is no theoretical justification for this learning-rate adjustment. Another drawback of the EG is that it requires storing all of the $\Omega(NT|Y|^{\kappa+1})$ dual variables for κ -th order CRFs, though not all of them may fit in memory for large problems. Note that the proposed method does not necessarily store the marginal probabilities for all of the N examples.

Algorithmic Differentiation (AD) can be used to evaluate the Hessian-vector products of CRFs, and requires $O(T|Y|^{\kappa+1})$ computations per instance (Pearlmutter 1994; Vishwanathan et al. 2006). AD is a general technique to compute derivatives using an algorithmic representation, i.e., program code. In computer programs, any function can be represented by a sequence of elementary operations. Based on the chain rule, the procedure of a derivative computation is evaluated by the sequence of the derivatives of the elementary operations. However, since AD requires all of the intermediate computational results of the gradient procedure to compute the Hessian-vector products, AD is resource intensive when trying to avoid the repetition of the gradient computations in the Newton-CG method.

Experiments

Experimental Settings

To assess the performance of the proposed method, we conducted experiments on two sequential labeling tasks: named entity recognition (NER) and phrase chunking. For the NER and chunking experiments, we used the data sets provided for the shared tasks of CoNLL2002 (Tjong Kim Sang 2002) and CoNLL2000 (Tjong Kim Sang and Buchholz 2000), respectively. For example, the NER shared task concentrates on four types of named entities: *persons*, *locations*, *organizations*, and *names of miscellaneous* entities. The words in the corpus are annotated with nine target labels, the beginning and continuation of the named entities and non-named entities. Figure 1 shows an example of named entity labels encoded in the data. The phrase chunking task is similar to the NER task, but the target labels are the beginning and continuation of the phrases such as noun phrases, verb phrases, and prepositional phrases. In addition, since the number of labeled samples is limited in some real-world tasks, we used

Task	Train	Dev.	Test	$ X $	$ Y $
NER	8,322	1,914	1,516	99,135	9
Chunking	8,936	N/A	2,012	338,539	23

Table 1: Statistics of the corpora.

the $1/4$ sized phrase chunking data as a small-sized problem.

We used a linear-chain CRF with the first order Markov model. The hyper-parameter σ of the CRF objective function f is selected using the development set or a subset of the training data. The features for the NER task are the same as the S3 features in (Altun, Johnson, and Hofmann 2003), and the features for the phrase chunking task are the same as the features provided by CRF++ package¹. The information for the corpora, the features, and the labels is summarized in Table 1.

We implemented three batch algorithms:

- NCG($c=\{0, 4K, ALL\}$): Newton-CG method using the proposed procedure with varying the number of sentences c for which the marginal probabilities are stored,
- LBFSG($m=\{5, 10, 50\}$): LBFSG storing $2m$ vectors and combined with a line search strategy and strong Wolf conditions (Nocedal and Wright 2006), and
- NCG(AD): Newton-CG method using algorithmic differentiation for the Hessian-vector products.

We also implemented three online algorithms:

- SGD($1/k$): SGD with time-varying learning-rates $\eta^k = \frac{\eta_0}{1+k/N}$ (Collins et al. 2008),
- SGD(α^k): SGD with time-varying learning-rates updated by $\eta^k = \eta_0 \alpha^{k/N}$ where we set $\alpha = 0.85$ as one of the suggested values in (Tsuruoka, Tsujii, and Ananiadou 2009), and
- EG: EG with the same initial value and the same update rule for the learning-rate as in (Collins et al. 2008).

The tuning parameter η^0 for SGDs is selected from $\{0.5, 0.1, 0.05, 0.01\}$ using the first 1,000 samples (500 for training and 500 for evaluation). In addition, for efficient sparse updates, the parameter vector of SGD is implemented as $\theta = av$ where a is a scalar and v is a dense vector (Shalev-Shwartz, Singer, and Srebro 2007).

For each epoch, the performance was evaluated according to the difference from the minimum function value on the training sets² and the standard F1 score on the test sets³. The former measures the quality of the solution in terms of optimization, and the latter measures the testing performance of predictive models. The stopping condition of the batch algorithms is the infinity norm of the gradient, $\|g_k\|_\infty \leq 0.05$. For the online algorithms, we simply stopped at $k = 200N$.

¹<http://crfpp.sourceforge.net/>

²We used the smallest function values of the systems as the approximation of the optimal function values.

³The F1 score is the harmonic mean of the precision and recall of entities or chunks, where the precision is the percentage of found named entities or chunks that are correct and the recall is the percentage of found named entities or chunks present in the corpus.

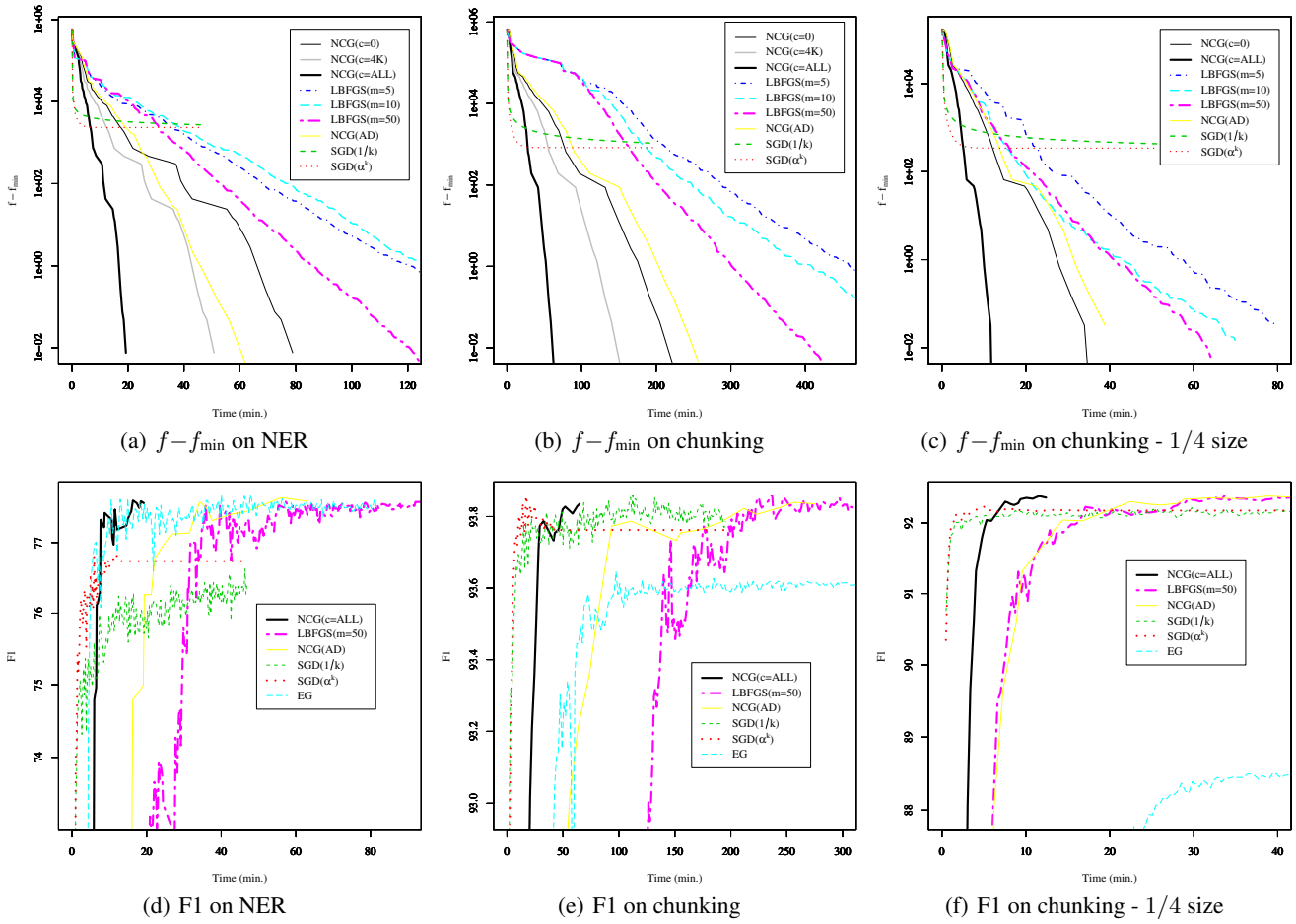


Figure 2: Difference from minimum function values with log-scale (a)–(c) and F1 scores (d)–(f) on the NER and chunking tasks

All of the algorithms were implemented using the same Java™ code base and ran on Java HotSpot™ 64-Bit Server VM (1.6.0_18). We used a Linux™ server with an Intel® Core™ 2 Quad 2.4 GHz CPU and 8 GB of memory.

Experimental Results

First, we report the convergence speeds of the function values in Figures 2(a), (b), and (c). All of the primal function values of the EG are omitted because they are relatively too large to show as lines in the figures. In addition, the result of $\text{NCG}(c=4K)$ is omitted for the 1/4 sized chunking data set in the sequel since the number of the training sentences is less than 4,000.

Overall, the proposed method, $\text{NCG}(c=\text{ALL})$, is the most accurate and fast optimization algorithm. $\text{NCG}(c=\text{ALL})$ is about 2–3 times faster than $\text{NCG}(c=0)$ which do not reuse the marginal probabilities. Compared with LBFGS, although both LBFGS and NCG methods are fastest when the most memory is used, the NCG method with $c=0$ is even faster than LBFGS with $m=50$. The adaptive quality control of the Newton step may be one of the reason why the NCG is superior to LBFGS which fixes the number of

Data set	batch method	online method
NER	$\text{NCG}(c=\text{ALL})$	EG
Chunking	$\text{NCG}(c=\text{ALL})$	$\text{SGD}(\alpha^k)$
Chunking - 1/4 size	$\text{NCG}(c=\text{ALL})$	$\text{SGD}(\alpha^k)$

Table 2: The fastest methods for each task: No online method perform the best in all of the tasks

vectors for the inverted Hessian approximation. Note that LBFGS with $m > 50$ could not fit in memory in the chunking task. $\text{NCG}(c=\text{ALL})$ also outperforms NCG with algorithmic differentiation, $\text{NCG}(\text{AD})$. This is because the proposed method can avoid the repeating the time-consuming computations in the CG iteration. The decreasing speed of f of the SGD methods is faster than $\text{NCG}(c=\text{ALL})$ at the beginning. However, the improvement of the function values is soon stalled. In contrast, the batch algorithms find accurate solutions because of their superlinear convergence properties.

Next, Figures 2(d), (e), and (f) show the comparisons in terms of the F1 scores on the test sets. We only show the best

results for the proposed method and LBFGS: NCG($c=ALL$) and LBFGS($m=50$). In terms of the speed to achieve the high testing performance, the proposed method is 3–6 times faster than LBFGS, and 2 times faster than NCG(AD). The results of the online algorithms are mixed, and no one performs best in all of the tasks. Table 2 shows the list of the best batch and online methods for each task. In addition, all of the online algorithms do not achieve the best testing performance on any of the data sets. The EG is the fastest algorithm for the NER task (Figure 2(d)), but the slowest algorithm on the chunking task (Figure 2(e) and 2(f)). $SGD(\alpha^k)$ is faster than $SGD(1/k)$ at the beginning because of the exponentially decaying learning-rate. However, it always converges to the suboptimal solutions. These task-dependent performances of the online algorithms may come from the deficiency of established learning-rate tuning methods.

Overall, the proposed method is faster than some of the online algorithms for all of the data sets. Since we never know the best online algorithm for each data set in advance, we can argue that the proposed method is the best overall.

Conclusion

For the linear-chain CRF learning problem, we propose a fast Newton-CG method employing a novel dynamic programming procedure for the Hessian-vector products. The proposed algorithm is suitable for Newton-CG methods in the sense that it can avoid the repetition of the time-consuming computations in the inner loop of the Newton-CG methods.

The sparse learning for CRFs is one of the interesting research directions. Recently, Newton-type algorithms were proposed for the supervised learning with sparsity-induced regularization (Schmidt et al. 2009; Kim, Sra, and Dhillon 2010; Tomioka et al. 2010). The proposed method may accelerate these algorithms when learning CRF models with the small number of non-zero parameters.

Acknowledgements

We thank Prof. S. V. N. Vishwanathan for providing the source code used in Vishwanathan et al. (2006).

References

Altun, Y.; Johnson, M.; and Hofmann, T. 2003. Investigating loss functions and optimization methods for discriminative learning of label sequences. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 145–152.

Bottou, L., and Bousquet, O. 2008. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, volume 20. 161–168.

Chen, M.; Chen, Y.; and Brent, M. R. 2008. CRF-OPT: An efficient high-quality conditional random field solver. In *Proceedings of 23rd AAAI Conference on Artificial Intelligence*, 1018–1023.

Collins, M.; Globerson, A.; Koo, T.; Carreras, X.; and Bartlett, P. L. 2008. Exponentiated gradient algorithms

for conditional random fields and max-margin Markov networks. *Journal of Machine Learning Research* 9:1775–1822.

Kim, D.; Sra, S.; and Dhillon, I. 2010. A scalable trust-region algorithm with application to mixed-norm regression. In *Proceedings of the 27th International Conference on Machine Learning*, 519–526.

Lafferty, J.; McCallum, A.; and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, 282–289.

Lin, C.-J.; Weng, R. C.; and Keerthi, S. S. 2008. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research* 9:627–650.

Nocedal, J., and Wright, S. J. 2006. *Numerical Optimization*. Springer-Verlag, second edition.

Pearlmutter, B. A. 1994. Fast exact multiplication by the Hessian. *Neural Computation* 6(1):147–160.

Schmidt, M.; van den Berg, E.; Friedl, M. P.; and Murphy, K. 2009. Optimizing costly functions with simple constraints: A limited-memory projected quasi-newton algorithm. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, 456–463.

Sha, F., and Pereira, F. 2003. Shallow parsing with conditional random fields. In *Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 134–141.

Shalev-Shwartz, S.; Singer, Y.; and Srebro, N. 2007. Pegasos: Primal Estimated sub-Gradient Solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, 807–814.

Spall, J. C. 2003. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley-Interscience.

Tjong Kim Sang, E. F., and Buchholz, S. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000*, 127–132.

Tjong Kim Sang, E. F. 2002. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, 155–158.

Tomioka, R.; Suzuki, T.; Sugiyama, M.; and Kashima, H. 2010. A fast augmented lagrangian algorithm for learning low-rank matrices. In *Proceedings of the 27th International Conference on Machine Learning*, 1087–1094.

Tsuruoka, Y.; Tsujii, J.; and Ananiadou, S. 2009. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th IJCNLP*, 477–485.

Vishwanathan, S. V. N.; Schraudolph, N. N.; Schmidt, M.; and Murphy, K. 2006. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the 23rd International Conference on Machine Learning*, 969–976.