Mach Learn (2009) 77: 195–224
DOI 10.1007/s10994-009-5142-6

Periodic step-size adaptation in second-order gradient descent for single-pass on-line structured learning

Chun-Nan Hsu · Han-Shen Huang · Yu-Ming Chang · Yuh-Jye Lee

Received: 28 April 2008 / Revised: 2 July 2009 / Accepted: 31 July 2009 /
Published online: 3 September 2009

Abstract It has been established that the second-order stochastic gradient descent (SGD) method can potentially achieve generalization performance as well as empirical optimum in a single pass through the training examples. However, second-order SGD requires computing the inverse of the Hessian matrix of the loss function, which is prohibitively expensive for structured prediction problems that usually involve a very high dimensional feature space. This paper presents a new second-order SGD method, called *Periodic Step-size Adaptation* (PSA). PSA approximates the Jacobian matrix of the mapping function and explores a linear relation between the Jacobian and Hessian to approximate the Hessian, which is proved to be simpler and more effective than directly approximating Hessian in an on-line setting. We tested PSA on a wide variety of models and tasks, including large scale sequence labeling tasks using conditional random fields and large scale classification tasks using linear support vector machines and convolutional neural networks. Experimental results show that single-pass performance of PSA is always very close to empirical optimum.

Keywords Conditional random fields · Convolutional neural networks · On-line learning · Sequence labeling · Stochastic gradient descent

Editors: Charles Parker, Yasemin Altun, and Prasad Tadepalli.

C.-N. Hsu (✉) · H.-S. Huang · Y.-M. Chang
Institute of Information Science, Academia Sinica, Taipei, Taiwan
e-mail: chunnan@iis.sinica.edu.tw

H.-S. Huang
e-mail: hanshen@iis.sinica.edu.tw

Y.-M. Chang
e-mail: porter@iis.sinica.edu.tw

Y.-J. Lee
Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan
e-mail: yuh-jye@mail.ntust.edu.tw

1 Introduction

The problem of large scale learning (Bottou and LeCun 2004, 2005; Bottou et al. 2007; Bottou and Bousquet 2008) arises partly due to the increasing of the digital data available for data mining and partly due to the need of many unsolved artificial intelligence challenges: natural language understanding, speech recognition, computer vision and robotics. In the latter case, *structured prediction* (Taskar 2005; Taskar et al. 2005; Tsochantaridis et al. 2005) is more adequate to deal with the complexity of these problems than well-studied binary classification methods, but then either a large number of examples or a set of very high dimensional training examples is required to achieve satisfactory generalization performance.

In this paper, we report the results of our investigation of how to apply a classical *on-line* learning method to solve the large scale structured prediction problem. Given a set of new training examples, an on-line learner can update its model without processing previously used examples while improving its performance. In contrast, *off-line* or batch learners must combine previous and new data and start the learning all over again. Otherwise, the performance may suffer.

The advantage of on-line learning for large scale structured prediction is illustrated in Fig. 1. Assume that everyday, a fixed number of training examples become available to the learner. Also assume that the time complexity for the off-line learner is no better than $O(n)$, where n is the data size. As days go by, the training examples will accumulate and the time required for the off-line learner to complete the learning task will eventually exceed the time required for the on-line learner to process those additional training examples. In fact, many well-designed on-line learning algorithms converge more rapidly than their off-line counterparts (see, for example, Collins et al. 2008; Bordes et al. 2007; Bottou et al. 2007; LeRoux et al. 2008) given the training examples of the same size.

For on-line learning to achieve the feat illustrated in Fig. 1 and solve large scale structured prediction problems in practical applications, an on-line learning algorithm should satisfy some desiderata. One of them is $O(d)$ or better per-iteration time complexity, where d is the dimension of the weights, so that very high dimensional data can be handled. Another is minimum batch size requirement. Some on-line learning algorithms need 100 or more training examples in each learning-iteration to cover potentially exponentially large output spaces. This places an unnecessary restriction to their applicability. Also, the number of hyper-parameters required to be tuned must be minimized.

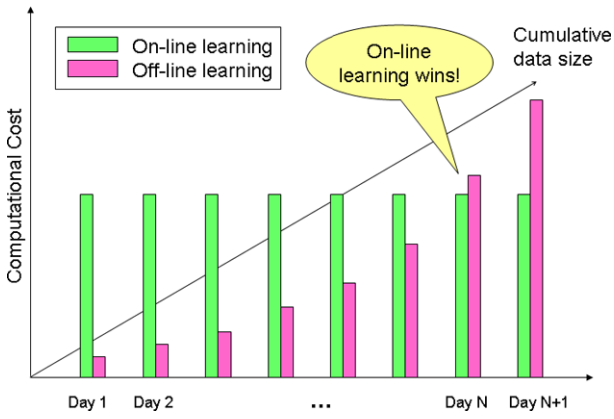


Fig. 1 Advantage of on-line learning

But above all, the most important is a minimum single-pass regret so that used training examples can be discarded to save time and storage. Previously, many authors, including Murata (1998), Murata and Amari (1999), Bottou and LeCun (2005), have established that the second-order stochastic gradient descent (SGD) method can potentially achieve generalization performance as well as empirical optimum in a single pass through the training examples. However, second-order SGD requires computing the inverse of the Hessian matrix of the loss function, which is prohibitively expensive, especially for structured prediction problems. This paper presents a new second-order SGD method, called *Periodic Step-Size Adaptation* (PSA). This new method satisfies all of the above mentioned desiderata for on-line methods. PSA approximates the Jacobian matrix of the mapping function and explores a linear relation given in (15) between the Jacobian and Hessian to approximate the Hessian, which is proved to be simpler and more effective than directly approximating Hessian in an on-line setting. We performed experiments to evaluate this new method on a wide variety of models and tasks, including large scale sequence labeling problems with conditional random fields (CRF) and classification tasks using linear support vector machines (SVM) and convolutional neural networks (CNN). Experimental results show that single-pass performance of PSA is always very close to empirical optimum. Since PSA is applicable to any learning problems where SGD is applicable, it may potentially be a practical solution of other large scale structured prediction problems.

This paper is organized as follows. Section 2 surveys related work in second-order stochastic gradient descent methods for on-line learning. Section 3 reviews a method for approximating the Jacobian matrix of a fixed-point iteration. From this approximation method, we derive PSA for on-line learning and analyze its convergence properties in Sect. 4. Section 5 reports our experimental results. Finally, we summarize our results and findings in Sect. 6.

2 Related work

Our new on-line learning algorithm is a second-order stochastic gradient descent method. Currently, most of state-of-the-art machine learning algorithms, including those for structured prediction problems, are derived by formulating a learning problem as an optimization problem to minimize an objective (loss) function $L(\theta; \mathsf{D})$ of a model where θ is the vector of the weights in the model and D is the set of training examples. To find the weights that minimize the objective function, we can solve this equation:

$$\frac{\partial L(\theta; \mathsf{D})}{\partial \theta} = \nabla L(\theta; \mathsf{D}) = \mathbf{g}(\theta; \mathsf{D}) = 0. \tag{1}$$

One of the methods to solve this equation is to use the gradient descent algorithm (Widrow and Hoff 1960), as illustrated in Fig. 2(a), where η is the step size. The step size can either be a scalar, a vector (to be multiplied with the gradient component-wise), or a matrix. Also shown in Fig. 2(b) is the stochastic gradient descent (SGD) algorithm. The difference between SGD and its off-line counterpart is the gradient taken in step 3. In the off-line version, the gradient is taken for the entire set of training examples D, while in SGD, only a small subset $\mathsf{B} \subseteq \mathsf{D}$, $|\mathsf{B}| \ll |\mathsf{D}|$, is considered at each iteration. SGD is applicable when the global objective function with regard to the entire set of training examples can be approximated as the sum of the objective function measured on independently drawn

(a) Gradient Descent (GD)

1: Initialize $\theta^{(0)}$
2: **repeat**
3: $\quad \theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \mathbf{g}(\theta^{(t)}; \mathsf{D})$
4: $\quad$ Update η
5: $\quad t \leftarrow t + 1$
6: **until** Convergence

(b) Stochastic Gradient Descent (SGD)

1: Initialize $\theta^{(0)}$
2: **repeat**
3: $\quad$ Choose $\mathsf{B}^{(t)}$ uniformly at random from D
4: $\quad \theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \mathbf{g}(\theta^{(t)}; \mathsf{B}^{(t)})$
5: $\quad$ Update η
6: $\quad t \leftarrow t + 1$
7: **until** Convergence

Fig. 2 (**a**) Batch (Off-Line) Gradient Descent Algorithm and (**b**) Stochastic (On-Line) Gradient Descent Algorithm

small batches:

$$L(\theta; \mathsf{D}) \approx \sum_{x \in \mathsf{D}} L(\theta; x) = \sum_{t=1}^{\frac{|\mathsf{D}|}{|\mathsf{B}|}} L(\theta; \mathsf{B}^{(t)}). \tag{2}$$

SGD has been an important algorithm for machine learning because it was applied to train back-propagating neural networks. Recently, Vishwanathan et al. (2006a) showed that by cleverly adjusting the step size in SGD, a large scale CRF model can be trained by scanning the training examples about 10 passes, while the best performing off-line algorithm requires several hundred passes. They also showed that their step size adjustment method, SMD, is applicable to large margin kernel methods for structured prediction (Vishwanathan et al. 2006b). Although SMD still takes longer CPU time to complete than the best off-line algorithm, their work demonstrated that on-line learning is promising for large scale structured prediction.

More recently, Bottou and Bousquet (2008) showed that well-tuned SGD can be very efficient for both SVM and CRF. In the case of SVMs with linear kernel, they showed that for a very large scale problem, widely used SVM packages, including LIBSVM (Chang and Lin 2001) and SVMlight (Joachims 1998, 2006), took hours to converge, but SGD only took less than 10 seconds. In fact, about ten years ago, LeCun et al. (1998b) and LeCun et al. (1999) already showed that SGD is feasible for training multiple-layered hierarchical models for large scale hand-written digit recognition and object recognition tasks. Their results lead the machine learning community re-consider SGD as a useful optimization algorithm and sheds new light on large scale structured prediction.

However, plain SGD and many of its variants still require scanning the training examples many passes to achieve the same level of generalization performance as their off-line counterparts. In fact, by appropriately adjusting the step size, a single pass through a large set of training examples is sufficient for SGD to reach the empirical optimum. To see why this is the case, consider solving (1) with Newton's method:

$$\theta^{(t+1)} = \theta^{(t)} - \mathbf{H}^{-1}(\theta^{(t)}) \nabla L(\theta^{(t)}; \mathsf{D}), \tag{3}$$

where $\mathbf{H}^{-1}$ is the inverse of the Hessian matrix of the objective function, defined by

$$\mathbf{H}(\theta) := \frac{\partial^2}{\partial \theta \partial \theta} L(\theta; \mathsf{D}).$$

Replacing η in step 3 of the algorithm in Fig. 2(a) with $\mathbf{H}^{-1}$ yields exactly the same equation. Based on Newton's method, Benveniste et al. (1990) showed that updating the step size with

the following equation is asymptotically optimal for SGD:

$$\eta_*^{(t)} = \frac{\mathbf{H}^{-1}(\theta^*)}{t+1}, \quad t \geq 0. \tag{4}$$

This is referred to as the second-order SGD (2SGD). Bottou and LeCun (2005) elaborated this result further, showing that given a sufficiently large n, the relation between the empirical optima given $n-1$ and n training examples is essentially the same Newton step given in (3): Once an empirical optimum for n training examples is reached, given an additional training example, a Newton step will bring the search to the empirical optimum for $n+1$ training examples. In other words, a single pass of 2SGD will generalize as well as empirical optimum.

However, computing the inverse of Hessian is prohibitively expensive even for a low dimensional problem. Alternatively, since many algorithms have been developed to approximate the Hessian for off-line learning, extending them for on-line learning could be a promising approach. One of the off-line algorithms that approximates the Hessian effectively is the L-BFGS algorithm (Nocedal and Wright 1999), which becomes well-known to machine learning community because of its efficiency for training large scale CRF models (Malouf 2002; Sha and Pereira 2003). An attempt was made to modify L-BFGS for on-line learning (Schraudolph et al. 2007). Their experimental result, however, showed that their new algorithm, oL-BFGS, requires 30 passes to train a CRF model for which their previous work SMD only needs about 7 passes. Though L-BFGS performs very well in off-line settings, it is challenging to modify L-BFGS for on-line learning because the approximation made by L-BFGS depends on the difference between gradient estimations in two consecutive iterations based on the entire set of training examples. In on-line settings, two consecutive gradients are obtained from two small batches. Their difference is no longer reliable for accurate approximation. Also, it is difficult to modify the line search used in L-BFGS to fit in on-line settings. A line search based on small batches usually leads to an inaccurate search direction, as in the case of conjugate gradient descent (Schraudolph and Graepel 2002). Instead, we should focus on how to minimize variance of Hessian approximation given that only a small batch of training examples is available at each iteration.

Other attempts of approximating the Hessian include diagonal approximation (Becker and LeCun 1988) and low rank approximation (LeCun et al. 1998b). Amari (1998) and more recently LeRoux et al. (2008) proposed stochastic algorithms based on the natural gradient, which can be proved to be asymptotically optimal, too. However, none of them is actually sufficient to achieve theoretical single-pass performance in practice. Compared to plain SGD, these 2SGD methods have almost no advantage in terms of computational cost (Bottou and Bousquet 2008).

The exponential gradient (EG) algorithm (Collins et al. 2008) is substantially different from stochastic gradient methods in that though they both process a single training example at a time, EG corresponds to block-coordinate descent in the dual, and uses the *exact* gradient with respect to the block being updated. As a result, EG is not a real on-line learning algorithm in the sense that EG cannot discard used training examples and learn incrementally in a single pass through the training examples as the scenario illustrated in Fig. 1. Moreover, when multiple-pass training is desirable, 2SGD is superior to EG because iterations required by EG is bounded by $O(n \log n)$ while 2SGD can reach an empirical optimum with $O(n)$ iterations.

3 Approximating Jacobian

This section reviews an efficient method to estimate the eigenvalues of the Jacobian in a fixed-point iteration mapping and an off-line learning method derived from this estimation method.

3.1 Jacobian of fixed-point mappings

Let θ be a d-dimensional vector in space $\mathbb{R}^d$ and $\mathcal{M}$ be a mapping $\mathcal{M} : \mathbb{R}^d \to \mathbb{R}^d$. The fixed-point iteration method solves the equation of the form $\theta = \mathcal{M}(\theta)$ by iteratively substituting the input of $\mathcal{M}$ with the output of $\mathcal{M}$ in the previous iteration:

$$\theta^{(t+1)} = \mathcal{M}(\theta^{(t)}), \qquad \theta^{(t+2)} = \mathcal{M}(\theta^{(t+1)}), \qquad \ldots.$$

The equation is solved when $\theta^{(t)}$ converges to θ^* with $\theta^* = \mathcal{M}(\theta^*)$. Many optimization algorithms, such as the EM algorithm (Dempster et al. 1977), can be formulated as a fixed-point iteration method.

Suppose that we apply a fixed-point iteration method from $\theta^{(t)}$ in the neighborhood of θ^* and the iteration converges at θ^*. Also suppose that the mapping $\mathcal{M}$ is differentiable. Then we can apply a linear Taylor expansion of $\mathcal{M}$ around θ^* so that

$$\theta^{(t+1)} = \mathcal{M}(\theta^{(t)}) \approx \theta^* + \mathbf{J}(\theta^{(t)} - \theta^*), \tag{5}$$

where $\mathbf{J}$ abbreviates $\mathcal{M}'(\theta^*)$, the Jacobian of the mapping $\mathcal{M}$ at θ^*. When $\lambda_i := \mathrm{eig}(\mathbf{J}) \in (-1, 1)$, the mapping is guaranteed to converge. If λ_i is positive, $\mathcal{M}$ converges monotonically along the i-th dimension; otherwise, $\mathcal{M}$ converges in an oscillating manner (Burden and Faires 1988), as illustrated in Fig. 3.

From (5), we have

$$\begin{aligned} \theta^{(t+1)} &\approx \theta^* + \mathbf{J}(\theta^{(t)} - \theta^*), \\ \theta^{(t)} &\approx \theta^* + \mathbf{J}(\theta^{(t-1)} - \theta^*) \\ \Rightarrow \theta^{(t+1)} - \theta^{(t)} &\approx \mathbf{J}(\theta^{(t)} - \theta^{(t-1)}). \end{aligned}$$

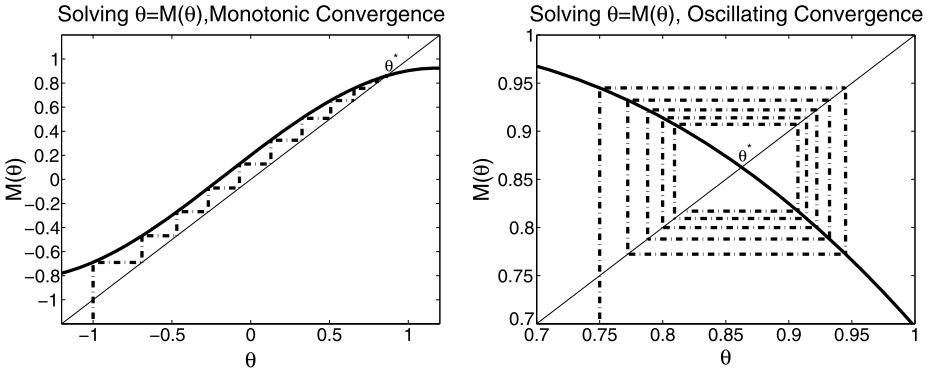


Fig. 3 In a univariate fixed-point iteration, the solution of $\theta = \mathcal{M}(\theta)$ is at the intersection of $y = \mathcal{M}(\theta)$ (*solid black curve*) and $y = \theta$ (*black diagonal line*). *Dash-dot line* tracks the progress of the convergence. When $0 < \mathcal{M}' < 1$, the fixed-point iteration converges monotonically (*left*). Otherwise, when $-1 < \mathcal{M}' < 0$, we have oscillating convergence (*right*)

Assume that there exists a scalar γ such that $\theta^{(t+1)} - \theta^{(t)} \approx \gamma(\theta^{(t)} - \theta^{(t-1)})$. Then we can approximate $\mathbf{J}$ by $\gamma\mathbf{I}$ with

$$\gamma := \frac{\|\theta^{(t+1)} - \theta^{(t)}\|}{\|\theta^{(t)} - \theta^{(t-1)}\|}. \tag{6}$$

We note that $\mathbf{I}$ denotes the identity matrix with the same size as $\mathbf{J}$.

In fact, γ is an approximation of λ_{max}, the largest eigenvalue of $\mathbf{J}$ (Schafer 1997; McLachlan and Krishnan 1997; Hesterberg 2005; Huang et al. 2005). The approximation will become exact as $t \to \infty$. This approximation is entailed from the rate of convergence of the EM algorithm, which is dependent on the largest eigenvalue of $\mathbf{J}$ (Dempster et al. 1977; Meng and Rubin 1994). The global rate of convergence of a fixed-point iteration method is defined as the ratio:

$$R := \lim_{t\to\infty} R^{(t)} \equiv \lim_{t\to\infty} \frac{\|\theta^{(t+1)} - \theta^*\|}{\|\theta^{(t)} - \theta^*\|}.$$

Dempster et al. (1977) showed that $R = \lambda_{max}$, the largest eigenvalue of $\mathbf{J}$ for the EM algorithm. Salakhutdinov and Roweis (2003) generalized this result to the family of bound optimization methods. In fact, it can be applied to any fixed-point iteration mapping. Since θ^* is unknown a priori, we replace it by empirical values to obtain γ as an estimate of λ_{max}. When t is sufficiently large, it can be expected that γ will be close to λ_{max}.

Similarly, the i-th component-wise rate of convergence is defined as (Meng and Rubin 1994):

$$R_i := \lim_{t\to\infty} R_i^{(t)} \equiv \lim_{t\to\infty} \frac{\theta_i^{(t+1)} - \theta_i^*}{\theta_i^{(t)} - \theta_i^*}.$$

We can estimate the i-th eigenvalue λ_i similarly by

$$\gamma_i := \frac{\theta_i^{(t+1)} - \theta_i^{(t)}}{\theta_i^{(t)} - \theta_i^{(t-1)}}, \quad \forall i. \tag{7}$$

One of the advantages of the estimate defined above is that it takes only $2d$ subtractions and a division to complete. Fraley (1999) empirically assessed the accuracy of the estimates given in (6) and (7) based on bivariate normal data with missing values and showed that both are reasonably good, especially when there are high percentages of missing data.

As a caveat, these estimates may fall out of the range or become numerically unstable either due to roundoff errors or to the fact that the iterates are not sufficiently close to a local optimum (Fraley 1999). Section 4.2 explains how to deal with these issues.

Previously, Hesterberg (2005) provided an informal proof that $\gamma \le \lambda_{max}$ from the eigen decomposition of $\mathbf{J}$. We can analyze γ_i similarly as follows. Let eigen decomposition $\mathbf{J} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}$ and $\mathbf{u}_i$ be column vectors of $\mathbf{Q}$ and $\mathbf{v}_i^T$ be row vectors of $\mathbf{Q}^{-1}$. Then we have

$$\mathbf{J}^t = \sum_{j=1}^{d} \lambda_j^t \mathbf{u}_j \mathbf{v}_j^T,$$

where λ_j is the j-th eigenvalue of $\mathbf{J}$. From (5),

$$\theta^{(t)} - \theta^* \approx \mathbf{J}^t(\theta^{(0)} - \theta^*),$$

$$\theta^{(t-1)} - \theta^* \approx \mathbf{J}^{t-1}(\theta^{(0)} - \theta^*)$$

$$\Rightarrow \Delta^{(t)} = \theta^{(t)} - \theta^{(t-1)} \approx \mathbf{J}^t\mathbf{J}^{-1}(\mathbf{J} - \mathbf{I})(\theta^{(0)} - \theta^*)$$

$$= \sum_{j=1}^{d} \lambda_j^t \mathbf{u}_j \mathbf{v}_j^T \mathbf{J}^{-1}(\mathbf{J} - \mathbf{I})(\theta^{(0)} - \theta^*)$$

$$\Rightarrow \Delta^{(t+1)} = \theta^{(t+1)} - \theta^{(t)} \approx \sum_{j=1}^{d} \lambda_j \lambda_j^t \mathbf{u}_j \mathbf{v}_j^T \mathbf{J}^{-1}(\mathbf{J} - \mathbf{I})(\theta^{(0)} - \theta^*).$$

Now let

$$w_{ij} := \mathbf{e}_i^T \mathbf{u}_j \mathbf{v}_j^T \mathbf{J}^{-1}(\mathbf{J} - \mathbf{I})(\theta^{(0)} - \theta^*),$$

where $\mathbf{e}_i$ is the i-th column of $\mathbf{I}$. Let Δ_i be the i-th element of Δ and λ_{j_i} be the largest eigenvalue of $\mathbf{J}$ such that $w_{ij} \neq 0$. Then

$$\gamma_i \equiv \frac{\Delta_i^{(t+1)}}{\Delta_i^{(t)}} = \frac{\sum_{j=1}^{d} \lambda_j^{t+1} w_{ij}}{\sum_{j=1}^{d} \lambda_j^t w_{ij}}$$

$$= \frac{\lambda_{j_i} + \sum_{j \neq j_i} (\lambda_j/\lambda_{j_i})^t \lambda_j w_{ij}/w_{ij_i}}{1 + \sum_{j \neq j_i} (\lambda_j/\lambda_{j_i})^t w_{ij}/w_{ij_i}}.$$

Therefore, we can conclude that

- $\gamma_i \to \lambda_{j_i}$ as $t \to \infty$ because $\forall i$, if $w_{ij} \neq 0$ then $\lambda_j/\lambda_{j_i} \leq 1$. $\lambda_{j_i} \equiv R_i$ is the i-th componentwise rate of convergence (Meng and Rubin 1994).
- $\gamma_i = \lambda_i$ if $\mathbf{J}$ is a diagonal matrix. In this case, our approximation is exact. This happens when there are high percentages of missing data for a Bayesian network model trained by EM (Hsu et al. 2006) and when features are uncorrelated for training a CRF model (Huang et al. 2009).
- γ_i is the average of eigenvalues weighted by $\lambda_j^t w_{ij}$. Since w_{ij} is usually the largest when $i = j$, we have $\gamma_i \approx \lambda_i$.

3.2 Aitken's acceleration

The estimates γ and γ_i have been applied in the framework of Aitken's acceleration to speed up the convergence of the EM algorithm (Hesterberg 2005; Huang et al. 2005; Hsu et al. 2006), suggesting that both γ and γ_i can be sufficiently accurate to allow for substantial speedup for the convergence of EM.

Let $\mathbf{J}$ be the Jacobian matrix of the fixed-point mapping $\mathcal{M}$ that we want to accelerate. The multivariate Aitken's acceleration is given by (McLachlan and Krishnan 1997):

$$\theta^{(t+1)} = \theta^{(t)} + (\mathbf{I} - \mathbf{J})^{-1}(\mathcal{M}(\theta^{(t)}) - \theta^{(t)}), \tag{8}$$

which is derived by successively applying (5) as follows:

$$\theta^* \approx \theta^{(t)} + \sum_{h=0}^{\infty} (\theta^{(t+h+1)} - \theta^{(t+h)})$$

$$\approx \theta^{(t)} + \sum_{h=0}^{\infty} \mathbf{J}^h(\theta^{(t+1)} - \theta^{(t)})$$
$$= \theta^{(t)} + (\mathbf{I} - \mathbf{J})^{-1}(\theta^{(t+1)} - \theta^{(t)}),$$

assuming that $\text{eig}(\mathbf{J}) \in (-1, 1)$ given that $\mathcal{M}$ will converge. Therefore, Aitken's acceleration can be considered as computing an extrapolation in an attempt to reach the solution θ^* in one step.

However, $\mathbf{J}$ may not have a closed form and computing $\mathbf{J}$ could be intractable even for a very simple model with a low dimensional weight space. A solution to this issue is to replace $(\mathbf{I} - \mathbf{J})^{-1}$ in (8) by the estimated eigenvalues of $\mathbf{J}$. Let $\mathbf{J} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$ be the eigen decomposition of $\mathbf{J}$ at θ^* and $\psi^* := \mathbf{Q}^{-1}\theta^*$ be the *eigen transformed* θ^*. Then in the eigenspace of $\mathbf{J}$, we have

$$\psi_i^* \approx \psi_i^{(t)} + \frac{1}{1 - \lambda_i}([\mathcal{M}(\psi^{(t)})]_i - \psi_i^{(t)}),$$

where λ_i is the i-th eigenvalue of $\mathbf{J}$. Then with the estimated eigenvalues $\gamma_i^{(t)}$ given in (7), we can perform extrapolation for each component by

$$\theta_i^{(t+1)} = \theta_i^{(t)} + (1 - \gamma_i^{(t)})^{-1}([\mathcal{M}(\theta^{(t)})]_i - \theta_i^{(t)}), \quad \forall i. \tag{9}$$

In this case, in order not to be confused, we replace $\theta_i^{(t+1)}$ in (7) by $\mathcal{M}(\theta^{(t)})_i$ and rewrite the definition of γ_i as

$$\gamma_i^{(t)} := \frac{[\mathcal{M}(\theta^{(t)})]_i - \theta_i^{(t)}}{\theta_i^{(t)} - \theta_i^{(t-1)}}, \quad \forall i. \tag{10}$$

It can be seen that to extrapolate to $\theta^{(t+1)}$, we need to apply $\mathcal{M}$ to $\theta^{(t-1)}$ consecutively to obtain $\gamma^{(t)}$. Huang et al. (2005) named this method as the *Triple Jump* extrapolation. This method can also be considered as a variant of *Steffensen* methods (Ortega and Rheinboldt 1970) in numerical analysis and is closely related to the *QuickProp* method (Fahlmann 1988) in neural networks. We will compare these related methods as well as PSA in Sect. 4.4.1.

3.3 Accelerating GIS with CTJPGIS

We present an application of the triple jump extrapolation to demonstrate that approximating Jacobian with (7) can be as effective for other fixed-point iteration methods as for EM. In this application, we try to accelerate generalized iterative scaling (GIS) for CRF training. GIS is a classical method for training exponential probabilistic models. When applied to large training sets for CRF, GIS is known to be prohibitively slow to converge (Malouf 2002; Sha and Pereira 2003). We derived a method called CTJPGIS, abbreviation of "*the component-wise triple jump method for penalized generalized iterative scaling*." We will show that CTJPGIS can accelerate GIS drastically, reducing the convergence time to a level that is comparable with L-BFGS for CRF training in terms of both rate of convergence and quality of trained models.

Let $\mathsf{D} := \{x_1, \ldots, x_n\}$ denote a set of n data sequences and $\{y_1, \ldots, y_n\}$ their corresponding labels. A CRF defines d features extracted from a given instance (x, y): $F(x, y) = (f_1(x, y), \ldots, f_d(x, y))^T$ and is parameterized by the weights for all features: $\theta = (\theta_1, \ldots, \theta_d)^T$. Training of CRFs is to search for the weight vector θ that minimizes the negative penalized log-likelihood function as the objective function. Usually we use a

Gaussian prior (Chen and Rosenfeld 1999) to avoid overfitting. Let $L(\theta; \mathsf{D})$ be the log-likelihood of D given θ. The penalized negative log-likelihood function $\mathcal{L}(\theta; \mathsf{D})$ is

$$\mathcal{L}(\theta; \mathsf{D}) = -L(\theta; \mathsf{D}) + \sum_i \frac{(\theta_i - \mu)^2}{2\sigma^2} + \text{const.} \tag{11}$$

Usually, μ is assigned to zero. Clearly, the objective function of CRF training satisfies (2) required for SGD to be applicable. The gradient of $\mathcal{L}$ along the direction of θ_i is

$$\nabla_i \mathcal{L}(\theta; \mathsf{D}) = Ef_i - \widetilde{E} f_i + \frac{\theta_i - \mu}{\sigma^2}, \tag{12}$$

where $\widetilde{E} f_i$ and Ef_i are empirical and model expectation of f_i, respectively.

To apply the triple jump extrapolation to the training of CRF, recall that the gradient for CRF is given in (12). Solving $\nabla_i \mathcal{L}(\theta; \mathsf{D}) = 0$ yields the penalized GIS (PGIS) mapping as follows:

$$\theta_i = \theta_i + \frac{1}{S} \log \frac{\widetilde{E} f_i}{Ef_i + \frac{\theta_i - \mu}{\sigma^2}}, \tag{13}$$

where $\frac{1}{S}$ is an arbitrary constant in $(0, 1)$ as the learning rate. Assigning $S := \max_{(x,y)\in \mathsf{D}} \times \sum_i f_i(x, y)$, the maximum number of feature occurrences in a training sequence, we obtain a new update rule, which is quite similar to the original GIS given in Lafferty et al. (2001). Assigning $\mathcal{M}(\theta)$ to be the RHS for (13) and applying the component-wise triple jump extrapolation described in (9), we have the CTJPGIS algorithm for training CRF.

We implemented CTJPGIS by replacing the L-BFGS optimization part in CRF++ (Kudo 2006).[1] We also ran CRF++ with default settings to obtain the performance results of L-BFGS. We used a tight termination condition

$$\left| \frac{\mathcal{L}(\theta^{(t-1)}) - \mathcal{L}(\theta^{(t)})}{\mathcal{L}(\theta^{(t)})} \right| < 10^{-7}$$

for all methods compared in this experiment, including L-BFGS, to ensure a fair comparison. The experiment was run on a Fedora 7 x86-64 Linux machine with AMD Athlon 64 X2 3800+ CPU and 4 GB RAM. Table 1[2] shows the tasks chosen for our comparison in this paper. The top four rows are sequence labeling tasks solved by CRF. These tasks have been

Table 1 Tasks for the experiments on sequence labeling

Task	Training	Test	Tag	Weight	Target	Reference
Base NP	8936	2012	3	1015662	94.0	(Sha and Pereira 2003)
Chunking	8936	2012	23	7448606	93.6	(Kudo 2006)
BioNLP/NLPBA	18546	3856	11	5977675	70.0	(Settles 2004)
BioCreative 2	15000	5000	3	10242972	86.5	(Hsu et al. 2008)

[1] Available under LGPL from the following URL: http://crfpp.sourceforge.net/.

[2] The target F-score for BioNLP/NLPBA is not >85%, as reported in Vishwanathan et al. (2006a), because it was due to a bug that included true labels as a feature, according to the author.

Table 2 Performance comparison of PGIS, CTJPGIS, and L-BFGS for CoNLL-2000, BioNLP/NLPBA-2004 and BioCreative 2 data sets

Data set		L-BFGS	CTJPGIS	PGIS
CoNLL-2000 Base NP	CPU Time (sec)	583	816	>41463
	Iteration	427	619	>30906
	Final F-score (%)	93.95	93.94	>93.35
BioNLP/NLPBA-2004	CPU time (sec)	63158	38800	>162462
	Iteration	1961	1279	>5161
	Final F-score (%)	70.33	70.26	>62.13
BioCreative 2	CPU time (sec)	4615	3011	>17656
	Iteration	895	639	>3926
	Final F-score (%)	86.77	86.49	>69.30

used in competitions and the performance was measured by F-score, which is the harmonic average of precision and recall:

$$p := \frac{TP \cdot 100\%}{TP + FP}, \qquad r := \frac{TP \cdot 100\%}{TP + FN}, \qquad F := \frac{2pr}{p + r},$$

where *TP*, *FP*, and *FN* are shorthands of true positives, false positives, and false negatives, respectively. The numbers of weights for CRF reported here are the values of "`Number of features`" produced by CRF++.

Table 2 shows the comparison of three methods: CTJPGIS, PGIS (the penalized GIS with Gaussian prior as given in (13)), and L-BFGS. The results show that CTJPGIS accelerates PGIS drastically and that CTJPGIS can compete with L-BFGS by winning in two out of three tasks in terms of both rate of convergence and CPU time. The achieved F-scores are all as good as those reported in the literature. The results suggest that the performance of approximating Jacobian by the triple jump extrapolation method is comparable to approximating Hessian by L-BFGS. These results provide empirical evidence that approximating Jacobian can be as effective as approximating Hessian in an off-line setting, suggesting that approximating Jacobian may be effective in an on-line setting as well.

4 Periodic step-size adaptation

The key issue of the 2SGD method is how to effectively approximate the Hessian matrix. We have presented a simple yet effective method to approximate the eigenvalues of the Jacobian matrix of a fixed-point iteration mapping. This section describes how this method can be applied to derive a new 2SGD method.

4.1 Approximating inverse of Hessian

In SGD, a batch is given at each iteration in an on-line setting and we apply

$$\theta^{(t+1)} = \theta^{(t)} - \eta \bullet \mathbf{g}(\theta^{(t)}; \mathsf{B}^{(t)}) \tag{14}$$

to update the weights at each iteration. The step size $\eta \in \mathbb{R}^d_+$ that we use is a positive vector-valued step size and "$\bullet$" denotes component-wise (Hadamard) product of two vectors.

From (4), the optimal step size is one that asymptotically approaches to $\mathbf{H}^{-1}$, the inverse of the Hessian matrix of $L(\theta^*; \mathsf{D})$. To avoid actually evaluating $\mathbf{H}^{-1}$, we can approximate $\mathbf{H}^{-1}$ with its eigenvalues. Considering a SGD iterate $\mathcal{M}(\theta) = \theta - \eta \bullet \mathbf{g}_{\mathsf{B}}(\theta)$, where $\mathbf{g}_{\mathsf{B}}$ is now re-written as a stochastic function depending only on θ. Clearly, $\mathcal{M}$ is a fixed-point iteration mapping, though a stochastic one, strictly speaking. Taking partial derivative of $\mathcal{M}$ with respect to θ, we have

$$\mathbf{J} = \mathcal{M}' = \mathbf{I} - \operatorname{diag}(\eta)\mathbf{H}. \tag{15}$$

By exploiting this linear relation between Jacobian and Hessian, we can approximate the inverse of Hessian by approximating Jacobian. Since

$$\operatorname{eig}(\mathbf{I} - \operatorname{diag}(\eta)\mathbf{H}) = \operatorname{eig}(\mathcal{M}') = \operatorname{eig}(\mathbf{J}) \approx \gamma,$$

where γ is an estimated eigenvalue of $\mathbf{J}$ as given in (7). When $\mathbf{H}$ is a symmetric matrix, its eigenvalue is given by

$$\begin{aligned} \operatorname{eig}(\mathbf{J}) &= 1 - \eta_i \operatorname{eig}(\mathbf{H}) \\ \Rightarrow \operatorname{eig}(\mathbf{H}) &= \frac{1 - \operatorname{eig}(\mathbf{J})}{\eta_i}. \end{aligned}$$

Therefore,

$$\operatorname{eig}(\mathbf{H}^{-1}) = \frac{\eta_i}{1 - \operatorname{eig}(\mathbf{J})} \approx \frac{\eta_i}{1 - \gamma_i}, \tag{16}$$

which implies that we can update the step size component-wise by

$$\eta_i^{(t+1)} \propto \frac{\eta_i^{(t)}}{1 - \gamma_i^{(t)}}. \tag{17}$$

If the mapping $\mathcal{M}$ converges, $\operatorname{eig}(\mathbf{J}) \in (-1, 1)$ and we can guarantee that the estimated $\operatorname{eig}(\mathbf{H}^{-1}) > 0$, which is required for the single pass result of SGD to hold (Bottou and LeCun 2005). A complete update rule for the step size will also involve a decay factor to ensure that the step size will approach zero.

4.2 Stability consideration

Due to the stochastic nature of small batch mapping, it is unlikely that we can obtain a reliable eigenvalue estimation at each single iteration. To increase stationary of the mapping, we take advantage of the law of large numbers and aggregate consecutive SGD mappings into a new mapping

$$\mathcal{M}^b = \underbrace{\mathcal{M}(\mathcal{M}(\ldots\mathcal{M}(\theta)\ldots))}_{b}.$$

Assume that the step size η is sufficiently small such that

$$\|\mathbf{g}(\theta^{(t)}; \mathsf{B}^{(t)}) - \mathbf{g}(\theta^{(t-1)}; \mathsf{B}^{(t)})\| \ll \|\mathbf{g}(\theta^{(t)}; \mathsf{B}^{(t)})\|.$$

Then at iteration t, applying SGD with a fixed step size η for b times yields

$$\begin{aligned}\theta^{(t+b)} &= \mathcal{M}^b(\theta^{(t)}) \\ &= \theta^{(t)} - \eta \sum_{i=0}^{b-1} \mathbf{g}(\theta^{(t+i)}; \mathsf{B}^{(t+i)}) \\ &\approx \theta^{(t)} - \eta \sum_{i=0}^{b-1} \mathbf{g}(\theta^{(t)}; \mathsf{B}^{(t+i)}) \\ &= \theta^{(t)} - b\eta \frac{1}{b} \sum_{i=0}^{b-1} \mathbf{g}(\theta^{(t)}; \mathsf{B}^{(t+i)}),\end{aligned}$$

which reduces the variance of gradient estimation by $\frac{1}{b}$, compared to the plain SGD mapping $\mathcal{M}$. The approximation is valid because $\theta^{(t+i)}, i = 0, \ldots, b-1$ are approximately fixed when η is sufficiently small (Benveniste et al. 1990).

We can proceed to estimate the eigenvalues of $\mathcal{M}^b$ from $\theta^{(t)}$, $\theta^{(t+b)}$ and $\theta^{(t+2b)}$. The estimation can be either by (6) to obtain a scalar-valued γ, or by (7) for each component to obtain a vector-valued γ. As Bottou and LeCun (2005) suggested, a full-rank approximation is necessary to achieve theoretical single-pass performance. Therefore, we apply (7) to estimate the eigenvalues of $\mathcal{M}^b$ for each component i:

$$\bar{\gamma}_i^b = \frac{\theta_i^{(t+2b)} - \theta_i^{(t+b)}}{\theta_i^{(t+b)} - \theta_i^{(t)}}. \tag{18}$$

A large b may appear to increase the stationary of the mapping $\mathcal{M}^b$ and improve the accuracy of our eigenvalue estimation. However, a too large b is inappropriate because in that case, the assumption that $\theta^{(t+i)}$ is approximately fixed will become invalid. We also note that our aggregate mapping $\mathcal{M}^b$ is different from a mapping that takes b small batches as the input in a single iteration. Their difference is similar to that between batch and stochastic gradient descent. Aggregate mappings have b chances to adjust its search direction, while mappings that use b small batches together only have one. Section 5.1.4 reports our experimental study of the impact of b to the performance of PSA.

With the estimated eigenvalues, we can present the complete update rule to adjust the step size vector η. To ensure that the estimated values of $\mathrm{eig}(\mathbf{J}) \in (-1, 1)$ and to ensure numerical stability, we introduce a positive constant $\kappa < 1$ as the upper bound of $|\bar{\gamma}_i^b|$. Let $\mathbf{u}$ denote the constrained $\bar{\gamma}^b$. Its components are given by

$$u_i := \mathrm{sgn}(\bar{\gamma}_i^b) \min(|\bar{\gamma}_i^b|, \kappa), \quad \forall i. \tag{19}$$

Then we can update the step size every $2b$ iterations based on $\mathbf{u}$ by:

$$\eta^{(t+2b+1)} = \mathbf{v} \bullet \eta^{(t+2b)}, \tag{20}$$

where $\mathbf{v}$ is a discount factor with components defined by

$$v_i := \frac{m + u_i}{m + \kappa + n}, \quad \forall i. \tag{21}$$

The discount factor is derived from (17) and the fact that when $u < 1$, $\frac{1}{1-u} > e^u \approx 1 + u$ to ensure numerical stability, with m and n controlling the range. Let α be the maximum value and β be the minimum value of v_i. We can obtain m and n by solving $\beta \le v_i \le \alpha$ for all i.

Since $-\kappa \leq u_i \leq \kappa$, we have $v_i = \alpha$ when $u_i = \kappa$ and $v_i = \beta$ when $u_i = -\kappa$. Solving these equations yields:

$$m = \frac{\alpha + \beta}{\alpha - \beta}\kappa \quad \text{and} \quad n = \frac{2(1-\alpha)}{\alpha - \beta}\kappa. \tag{22}$$

For example, if we want to set $\alpha = 0.9999$ and $\beta = 0.99$, then m and n will be 201κ and 0.0202κ, respectively. Setting $0 < \beta < \alpha \leq 1$ ensures that the step size is decreasing and approaches zero so that its convergence can be guaranteed (Benveniste et al. 1990; Spall 2003).

We informally describe the behavior of our step size update rule from the view point of fixed-point iteration. We note that the step size will always be reduced after every update in our setting. But according to (16), when the estimated eigenvalue $\bar{\gamma}_i^b$ is positive, SGD converges monotonically to the optimum. In this case, v_i will be large, or the reduction will be less, to maintain a large step size along the i-th dimension. On the other hand, when $\bar{\gamma}_i^b$ is negative, the convergence is oscillating and v_i will be small to reduce the step size further. Figure 3 illustrates different types of convergence dictated by the sign of the eigenvalue. The scale of the reduction is controlled to be proportional to the estimated eigenvalues of $\mathbf{H}^{-1}$ in an attempt to achieve theoretical single-pass performance.

In practice, we found that assigning α, β and κ to values within the following ranges is sufficient to produce adequate convergence performance:

- $\alpha : [0.999, 0.9999]$,
- $\beta : [0.96, 0.99]$,
- $\kappa : [0.9, 0.99]$.

That leaves b as the only additional parameter that requires careful tuning. In Sect. 5.1.4, we will report an empirical study of the impact of b on the convergence and provide a practical guideline to specify these parameters.

4.3 The PSA algorithm

Algorithm 1 shows the PSA algorithm. In a nutshell, PSA applies SGD with a fixed step size and periodically updates the step size by approximating Jacobian of the aggregated mapping. The complexity per iteration is $O(\frac{d}{b})$ because the cost of eigenvalue estimation given in (18) is $2d$ and it is required for every $2b$ iterations. For models with a regularization term, such as CRF, we can apply a technique that represents a weight vector as the product of a scalar and a vector to further reduce the per iteration cost (see, Bottou and Bousquet 2008). We will discuss this in details in Sect. 5.1.1.

4.4 Analysis of PSA

We first compare PSA, CTJPGIS and their related methods in the literature and then present an analysis of PSA's asymptotic convergence property.

4.4.1 Comparison with QuickProp

PSA can be considered as a member in the family of the *discretized Newton methods* in the numerical analysis literature (Ortega and Rheinboldt 1970). When the problem is to solve a system of nonlinear equations $\mathbf{g}(\theta) = 0$, the first order optimality condition of an optimization problem, the general form of these methods is

$$\theta^{(t+1)} = \theta^{(t)} - \mathbf{A}[\theta, h]^{-1}\mathbf{g}(\theta^{(t)}),$$

Algorithm 1 The PSA Algorithm

Given: α, β, $\kappa < 1$ and b
Initialize $\theta^{(0)}$ and $\eta^{(0)}$; $t \leftarrow 0$
$m \leftarrow \frac{\alpha+\beta}{\alpha-\beta}\kappa$ and $n \leftarrow \frac{2(1-\alpha)}{\alpha-\beta}\kappa$ ▷ Equation (22)
repeat
  Choose a small batch $\mathsf{B}^{(t)}$ uniformly at random from the set of training examples D
  update $\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \bullet \mathbf{g}(\theta^{(t)}; \mathsf{B}^{(t)})$ ▷ Equation (14) i.e., SGD update
  if $(t+1) \bmod 2b = 0$ **then** ▷ Update η
    update $\bar{\gamma}_i^b \leftarrow \frac{\theta_i^{(t+2b)} - \theta_i^{(t+b)}}{\theta_i^{(t+b)} - \theta_i^{(t)}}$ ▷ Equation (18)
    For all i, update $u_i \leftarrow \operatorname{sgn}(\bar{\gamma}_i^b)\min(|\bar{\gamma}_i^b|, \kappa)$ ▷ Equation (19)
    For all i, update $v_i \leftarrow \frac{m+u_i}{m+\kappa+n}$ ▷ Equation (21)
    update $\eta^{(t+1)} \leftarrow \mathbf{v} \bullet \eta^{(t)}$ ▷ Equation (20)
  else
    $\eta^{(t+1)} \leftarrow \eta^{(t)}$
  end if
  $t \leftarrow t+1$
until Convergence

where $\mathbf{A}$ is a matrix designed to approximate $\mathbf{H}$ without explicitly computing the partial derivatives to save computational time. Since full-matrix approximation is too costly, we can use a divided difference for each component:

$$\mathbf{A}[\theta, h] = \operatorname{diag}\left(\frac{\mathbf{g}_i(\theta) - \mathbf{g}_i(\theta - h)}{h_i}\right).$$

We discuss two well-known discretized Newton methods here: the secant method and the Steffensen method. Each of them has many variants (Ortega and Rheinboldt 1970). We will focus on their variants that are closely related to PSA and CTJPGIS discussed in Sect. 3.3.

When applied to accelerate gradient-descent, an instantiation of the multivariate secant method can be defined as the following iterates:

$$\theta^{(t+1)} = \mathcal{S}(\theta^{(t)}) := \theta^{(t)} - \mathbf{A}\left[\theta^{(t)}, \theta^{(t)} - \theta^{(t-1)}\right]^{-1}\mathbf{g}(\theta^{(t)}).$$

This method is also known as QuickProp (Fahlmann 1988) in the neural network literature. Its update rule for each component can be expressed as:

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \left[\frac{\theta_i^{(t)} - \theta_i^{(t-1)}}{\mathbf{g}_i(\theta^{(t)}) - \mathbf{g}_i(\theta^{(t-1)})}\right]\mathbf{g}_i(\theta^{(t)}), \quad \forall i. \tag{23}$$

The Steffensen method is a generalized form of (9) and (10). It is an improved version of Aitken's acceleration. When applied to a gradient-descent mapping, the update rule of the Steffensen method for each component is given by

$$\theta_i^{(t+1)} = \begin{cases} [\mathcal{M}(\theta^{(t)})]_i = \theta_i^{(t)} - \eta\mathbf{g}_i(\theta^{(t)}) & \text{if } t \bmod 2 = 0, \\ \theta_i^{(t+1)} = \theta_i^{(t)} - \left[\frac{\theta_i^{(t)} - \theta_i^{(t-1)}}{\mathbf{g}_i(\theta^{(t)}) - \mathbf{g}_i(\theta^{(t-1)})}\right]\mathbf{g}_i(\theta^{(t)}) & \text{otherwise.} \end{cases} \tag{24}$$

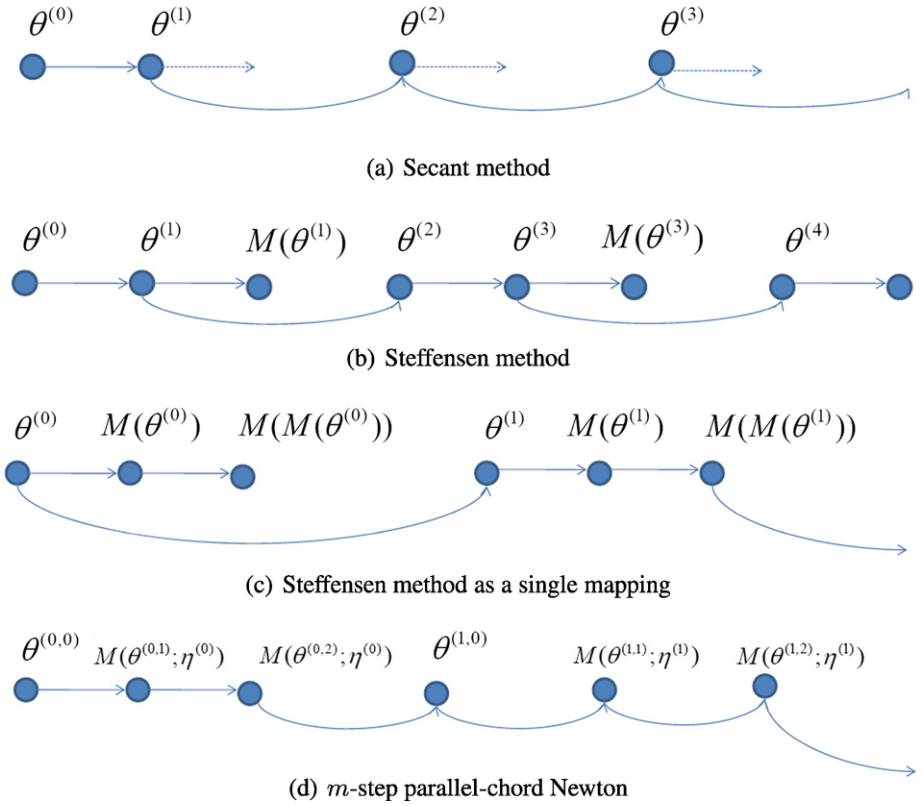


Fig. 4 Iterations of discretized Newton methods. A *straight line* indicates an application of $\mathcal{M}$, in our case the gradient-descent. A *curve line* indicates applying an approximated Newton step. A *dash line* in (**a**) indicates a gradient evaluation but no new weight vector is derived

The "otherwise" case turns out to be identical with the secant method. However, if we rewrite (24) as a single mapping rule, we obtain

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \left[\frac{\eta \mathbf{g}_i(\theta^{(t)})}{\mathbf{g}_i(\theta^{(t)}) - \mathbf{g}_i(\theta^{(t)} - \eta \mathbf{g}(\theta^{(t)}))}\right] \mathbf{g}_i(\theta^{(t)}),$$

which is equivalent to the following general form:

$$\theta^{(t+1)} = \theta^{(t)} - \mathbf{A}[\theta^{(t)}, \eta \mathbf{g}(\theta^{(t)})]^{-1} \mathbf{g}(\theta^{(t)}).$$

Figure 4(a)–(c) illustrates the progressions of the secant and Steffensen methods. We can see that they are indeed quite different. Both methods apply the same approximated Newton step (i.e., (23)), which approximates the tangent hyperplane with slope $\mathbf{H}$ by a secant $\mathbf{A}$ with respect to a pair of weight vectors. The secant method estimates $\mathbf{H}$ with respect to $\theta^{(t-1)}$ and $\mathcal{S}(\theta^{(t-1)})$, while the Steffensen method is based on $\theta^{(t-1)}$ and $\mathcal{M}(\theta^{(t-1)})$. The approximation made by the Steffensen method could be more accurate than that by the secant method because intuitively, the secant line measured from a pair of points is close to the tangent line if the pair are close together and it is likely that $\|\mathcal{M}(\theta^{(t-1)}) - \theta^{(t-1)}\| < \|\mathcal{S}(\theta^{(t-1)}) - \theta^{(t-1)}\|$.

In fact, it is well-known that the secant method and its variants have guaranteed local convergence of order $(1+\sqrt{5})/2 \approx 1.61$, while the Steffensen method and its variants have local quadratic convergence, which is as efficient as Newton (with exact evaluation of $\mathbf{H}^{-1}$). Given that all discretized Newton methods have an advantage that they need not to explicitly compute the partial derivatives to obtain $\mathbf{H}^{-1}$, they are promising candidates of approximated 2SGD. Among them, the Steffensen method is a better choice than the secant method because of its higher order of convergence.

PSA, however, is not a secant nor a Steffensen method. PSA is a variant of *"m-step parallel-chord Newton"* method (Ortega and Rheinboldt 1970, p. 187). A parallel-chord Newton method evaluates $\mathbf{H}^{-1}$ at $t=0$, and then uses the result for the rest of the iterations. A "m-step" parallel-chord Newton method re-evaluate $\mathbf{H}^{-1}$ periodically when $t \bmod m = 0$, which can be written as the following two-layered iteration:

$$\theta^{(k+1,0)} = \theta^{(k,m)}, \qquad \theta^{(k,t+1)} = \theta^{(k,t)} - \mathbf{H}^{-1}(\theta^{(k,0)})\mathbf{g}(\theta^{(k,t)}), \quad t = 0, \ldots, m-1.$$

Traub (1964) showed that the m-step parallel-chord Newton method with exact Hessians has local convergence of order $m+2$:

$$\|\theta^{(k+1)} - \theta^{(*)}\| \leq c\|\theta^{(k)} - \theta^{(*)}\|^{(m+2)}.$$

In 1967, Šamanskii (Ortega and Rheinboldt 1970, p. 365) has shown that under certain conditions, m-step parallel-chord Newton with discrete approximation $A(\theta, h)$ still converges locally and the order is $m+1$. Consequently, it has been established that a periodic update method with discrete approximation of $\mathbf{H}^{-1}$ in an off-line optimization setting not only save computational cost but can also achieve an excellent rate of convergence.

PSA can be considered as an on-line variant of the m-step parallel-chord Newton method with $m=3$ and each mapping is a composite mapping of $2b$ discrete Newton steps:

$$\theta^{(k+1,0)} = \theta^{(k,2b)}, \qquad \theta^{(k,t+1)} = \theta^{(k,t)} - \eta^{(k)} g(\theta^{(k,t)}), \quad t = 0, \ldots, 2b-1,$$

$$\eta^{(k+1)} = \frac{\eta^{(k)} G(\theta^{(k,0)})}{G(\theta^{(k;b)}) - G(\theta^{(k;0)})}, \qquad G(\theta^{(k,0)}) = \sum_{j=0}^{b-1} g(\theta^{(k,j)}).$$

Figure 4(d) illustrates this iteration. Therefore, if the estimation is sufficiently accurate, asymptotic behavior of PSA is supposed to be close to order 4.

4.4.2 Asymptotic convergence property

Previously, Murata (1998) has derived a general form of the expectation and variance of $\theta^{(t)}$ obtained by SGD. When we have the least possible step size $\eta^{(t+1)} = \beta\eta^{(t)}$ for all $t \bmod 2b = 0$ in PSA, the expectation of $\theta^{(t)}$ obtained by PSA can be shown to be:

$$E(\theta^{(t)}) = \theta^* + \prod_{k=1}^{t} \left(I - \eta^{(0)} \beta^{\lfloor \frac{k}{b} \rfloor} \mathbf{H}(\theta^*; \mathsf{D}) \right) (\theta^{(0)} - \theta^*) \tag{25}$$

$$= \theta^* + \mathbf{S}^{(t)}(\theta^{(0)} - \theta^*). \tag{26}$$

The rate of convergence is governed by the largest eigenvalue of $\mathbf{S}^{(t)}$. We now derive a bound of this eigenvalue.

Theorem 1 *Let λ_h be the least eigenvalue of $\mathbf{H}(\theta^*; \mathsf{D})$. The asymptotic rate of convergence of PSA is bounded by*

$$\mathrm{eig}(\mathbf{S}^{(t)}) \leq \exp\left\{\frac{-\eta^{(0)}\lambda_h b}{1-\beta}\right\}.$$

Proof We can show that

$$\begin{aligned}\mathrm{eig}(\mathbf{S}^{(t)}) &= \prod_{k=1}^{t}\left(1-\eta^{(0)}\beta^{\lfloor\frac{k}{b}\rfloor}\lambda_h\right)\\ &\leq \exp\left\{-\sum_{k=1}^{t}\eta^{(0)}\lambda_h\beta^{\lfloor\frac{k}{b}\rfloor}\right\} = \exp\left\{-\eta^{(0)}\lambda_h\sum_{k=1}^{t}\beta^{\lfloor\frac{k}{b}\rfloor}\right\}\end{aligned}$$

because for any $0 \leq a_j < 1$, since $1 - a_j \leq e^{-a_j}$,

$$0 \leq \prod_{j=1}^{n}(1-a_j) \leq \prod_{j=1}^{n} e^{-a_j} = e^{-\sum_{j=1}^{n} a_j}.$$

Now, since

$$\sum_{k=1}^{t}\beta^{\lfloor\frac{k}{b}\rfloor} \approx \left(\sum_{l=0}^{\lfloor\frac{t}{b}\rfloor} b\beta^l\right) = b\sum_{l=0}^{\lfloor\frac{t}{b}\rfloor}\beta^l \to \frac{b}{1-\beta} \quad \text{when } t\to\infty,$$

we have

$$\mathrm{eig}(\mathbf{S}^{(t)}) \leq \exp\left\{-\eta^{(0)}\lambda_h\sum_{k=1}^{t}\beta^{\lfloor\frac{k}{b}\rfloor}\right\} \to \exp\left\{\frac{-\eta^{(0)}\lambda_h b}{1-\beta}\right\} \quad \text{when } t\to\infty. \qquad \square$$

Though this analysis suggests that for rapid convergence to θ^*, we should assign $\beta \approx 1$ with a large b and $\eta^{(0)}$, it is based on a worst-case scenario and thus insufficient as a practical guideline for parameter assignment. Section 5.1.4 will provide an empirically derived guideline for the purpose.

5 Experimental results

This section reports the experimental results of applying PSA to a variety of models and large scale tasks, including CRF training for sequence labeling tasks, large scale training of linear SVM for binary classification, and multi-classification by a convolutional neural network model.

5.1 Conditional random fields

We compared PSA with plain SGD and SMD (Vishwanathan et al. 2006a) to evaluate PSA's performance for the four tasks as given in Table 1. Again, we implemented PSA by replacing the L-BFGS optimizer in CRF++. For SMD, we used the implementation available

in the public domain.[3] Our SGD implementation for CRF is from Bottou.[4] All the above implementations are revisions of CRF++. Finally, we ran the original CRF++ with default settings to obtain the performance results of L-BFGS. The parameter settings in our experiments are as follows:

- SGD: We simply keep the original settings intact.
- SMD: We used the typical setting described in Vishwanathan et al. (2006a), that is, $\mu = 0.1$, $\lambda = 1.0$, and $\eta^{(0)} = 0.1$. The batch size is set to 6 or 8 for different tasks as specified in Vishwanathan et al. (2006a).
- PSA: We used $\kappa = 0.9$, $(\alpha, \beta) = (0.9999, 0.99)$, $b = 10$, and $\eta_i^{(0)} = 0.1,\ \forall i$. The batch size is one for all tasks.

All of the experiments reported here for CRF were ran on an Intel Q6600 Fedora 8 i686 PC with 4 G RAM.

5.1.1 Single-pass performance comparison

Table 3 compares SGD variants in terms of the execution time and F-scores achieved after processing the training examples for a single pass. Since the loss function in CRF training is convex, the convergence results of L-BFGS can be considered as the empirical minimum. The results show that single-pass F-scores achieved by PSA are about as good as the empirical minima, suggesting that PSA has effectively approximated Hessian in CRF training. This is important because an on-line algorithm is useful when the training examples can be discarded after they are used.

In the BioCreative 2 task, PSA's single pass F-score is much better than the other two SGD variants, but still is about 6 percentage point below the empirical optimum. However, in less than two passes, PSA catches up to 85 and in 4 passes reaches 86.46. Single-pass F-scores of plain SGD and SMD are way below empirical optima. We note that for BioCreative 2, the F-score achieved by plain SGD in a single pass is much lower than other methods. In fact, plain SGD reached 68.25 after 0.8 passes but dropped sharply to 34.33 when it finished one pass. This is because its learning curves fluctuate wildly, as shown in Fig. 5.

In terms of the CPU time, though as expected, plain SGD is the fastest, it is remarkable that PSA is faster than SMD for all tasks. SMD is supposed to have an edge here because the mini-batch size for SMD was set to 6 or 8, as specified in Vishwanathan et al. (2006a), while

Table 3 CPU time in seconds and F-scores achieved after a single pass for CRF training for SGD variants and after convergence for L-BFGS

Method (pass)	Base NP		Chunking		BioNLP/NLPBA		BioCreative 2	
	time	F-score	time	F-score	time	F-score	time	F-score
SGD (1)	1.15	92.42	13.04	92.26	12.23	66.37	3.18	34.33
SMD (1)	41.50	91.81	350.00	91.89	522.00	66.53	497.71	69.04
PSA (1)	16.30	93.31	160.00	93.16	206.00	69.41	191.61	80.79
L-BFGS (batch)	221.17	93.91	8694.40	93.78	20130.00	70.30	1601.50	86.82

[3] Available under LGPL from the following URL: http://sml.nicta.com.au/code/crfsmd/.

[4] http://leon.bottou.org/projects/sgd.

PSA used one for all tasks, implying that PSA must perform many times more gradient-descent iterates than SMD in a pass. But PSA is still faster than SMD partly because PSA can take advantage of *the sparsity trick* as plain SGD (Shalev-Shwartz et al. 2007).

For very high dimensional models, variants of SGD could be slow due to the need to update the entire vector for every small batch in the set of training examples. But for SGD, the sparsity trick can be applied to drastically reduce the cost of updating. In the case of CRF training, given a small batch $\mathsf{B}^{(t)}$ in a sparse data set, it is quite likely that the expected value of the i-th feature is zero. In that case, we have $\nabla_i \mathcal{L}(\theta^{(t)}, \mathsf{B}^{(t)}) = \frac{\theta_i^{(t)}}{\sigma^2}$ and we can update $\theta_i^{(t)}$ by

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \eta^{(t)} \frac{\theta_i^{(t)}}{\sigma^2} = \theta_i^{(t)} \left(1 - \frac{\eta^{(t)}}{\sigma^2}\right).$$

That is, most weights can be updated by multiplying $(1 - \frac{\eta^{(t)}}{\sigma^2})$.

SGD can take advantage of this trick by decomposing θ to $s\phi$, where s is a scaling factor and ϕ a vector. Then, the weight update is performed based on the expected occurrences of the features as follows:

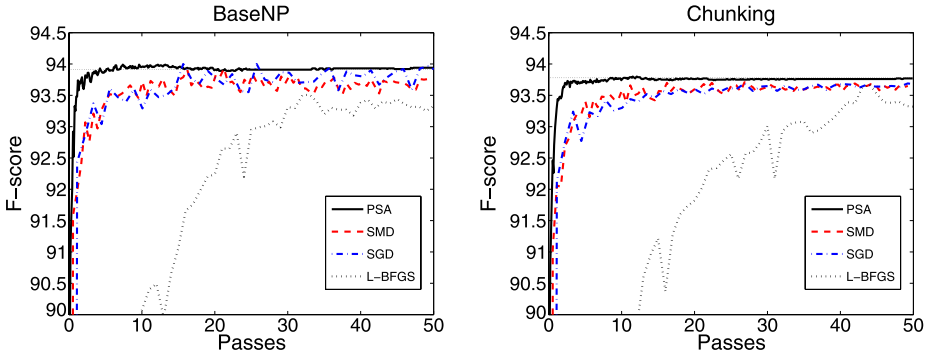
1. $s^{(t+1)} = s^{(t)}(1 - \frac{\eta^{(t)}}{\sigma^2})$;
2. if the expected occurrence of feature i is 0, ϕ_i remains unchanged;
3. otherwise, $\phi_i^{(t+1)} = (s^{(t)}\phi_i^{(t)} - \eta^{(t)}\nabla_i \mathcal{L}(\theta^{(t)}, \mathsf{B}^{(t)}))/s^{(t+1)}$.

The sparsity trick can be applied to PSA in almost the same manner during every $2b$ iterates before updating the step size. However, it is not applicable to SMD because SMD performs a local step-size updates independently at every iteration. This could be one of the reasons why the batch size for SMD must be set to more than one to avoid intensive weight updates. PSA can save more time if the data is very sparse, as more features will have zero expected occurrence in $\mathsf{B}^{(t)}$. The chance usually decreases if we have more labels in the model, as in the case of the CoNLL 2000 chunking task, where 23 labels are used.

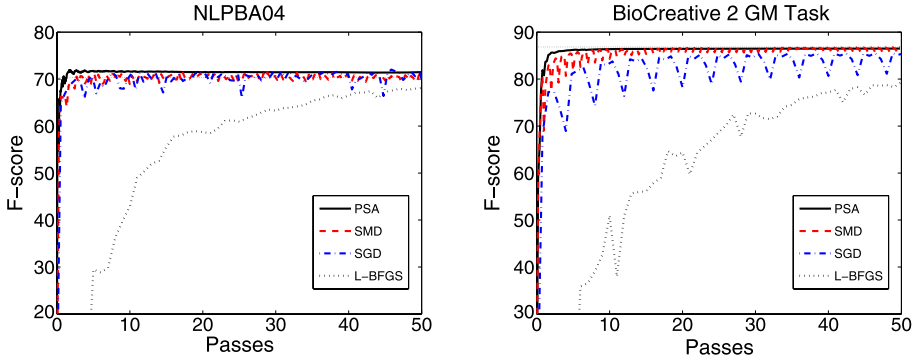
5.1.2 Convergence performance comparison

Figure 5 shows the learning curves of all methods for the four chosen tasks. The learning curves plot the progress of test-set F-scores as a function of the number of processed examples, measured by the number of passes through the entire training data set. We reported the learning curves for only the first 50 passes because by then all on-line methods have already converged, though L-BFGS would still require many more passes to converge. As expected, the learning curves show that PSA clear outperformed all methods in terms of the number of passes, with its F-scores approaching the optimum immediately after one pass. The superiority of PSA becomes even more obvious for the two biological entity recognition tasks that are more challenging. Both of the tasks require a large set of features. The fluctuation of plain SGD is notable in all tasks, especially for BioCreative 2.

Figure 6 also shows the learning curves but they plot the progress of training set objective (loss) values as a function of passes. Again, PSA outperformed all other methods, while SGD is the slowest and has the widest fluctuation among all SGD variants. However, though its progress appears to stall at a much higher objective value than other methods, it seems that its F-scores were not suffered and remain only slightly lower than those of PSA and SMD after 20 passes.



(a) CoNLL 2000 base NP and chunking tasks



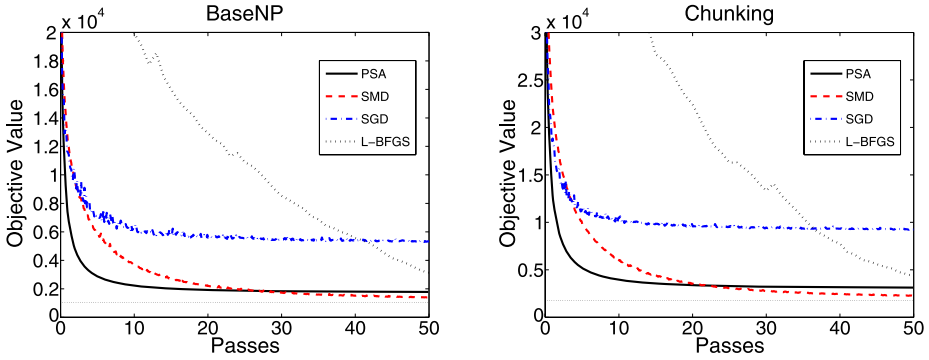
(b) BioNLP/NLPBA and BioCreative 2

Fig. 5 Comparing convergence performance in terms of F-scores by PSA, SGD, SMD and L-BFGS for four sequence labeling tasks with CRF

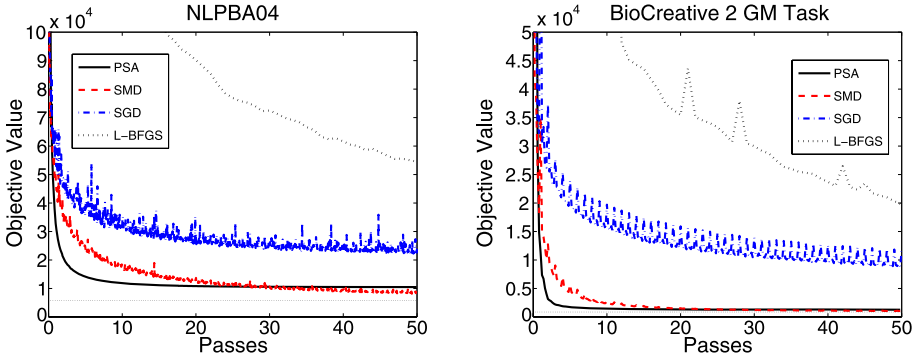
5.1.3 Execution time comparison

We also compare the CPU time required to reach a certain F-score for these SGD variants. Figure 7 shows the learning curves with respect to the CPU time. In the beginning, plain SGD appears to be the fastest but as it flattens out, PSA quickly catches up, usually immediately after scanning one pass of the training examples. Similar results can be observed from the learning curves of objective values with respect to the CPU time, given in Fig. 8.

We also performed a bootstrap test to compare the performance of single-pass PSA and other methods given the same CPU time spent. That is, we stopped all methods when they had run for as much CPU time as single-pass PSA. Then we used bootstrap resampling on the test set for each task. For 1,000 trials, a random sample of the same size as the test set was selected *with replacement* from the test set and the F-score was computed using the selected sample for the model obtained by each method. Single-pass PSA is statistically significantly better than another method if the proportion of the times that in these 1,000 trials, the F-score by PSA exceeded the F-score of that method is greater than 95%, and vice versa. Table 4 gives the results, which show that single-pass PSA outperforms SMD in all tasks and L-BFGS in three out of four tasks with statistical significance. Single-pass PSA's performance is comparable with plain SGD given the same CPU time, but PSA has the advantage that it only needs one pass through the training examples.



(a) CoNLL 2000 base NP and chunking tasks



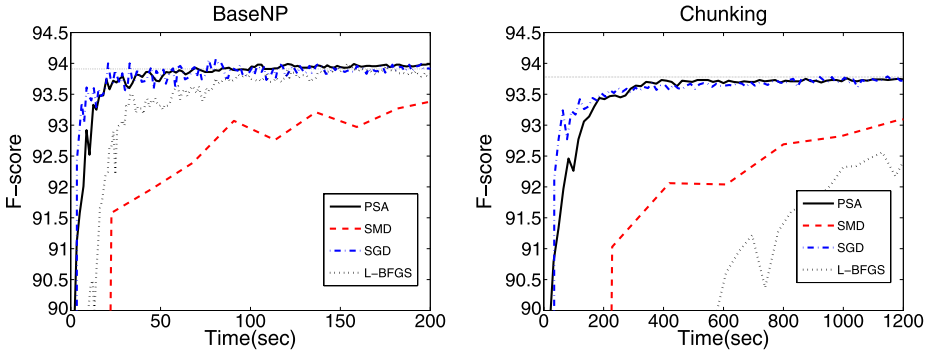
(b) BioNLP/NLPBA and BioCreative 2

Fig. 6 Comparing convergence performance in terms of objective (loss) values by PSA, SGD, SMD and L-BFGS for four sequence labeling tasks with CRF

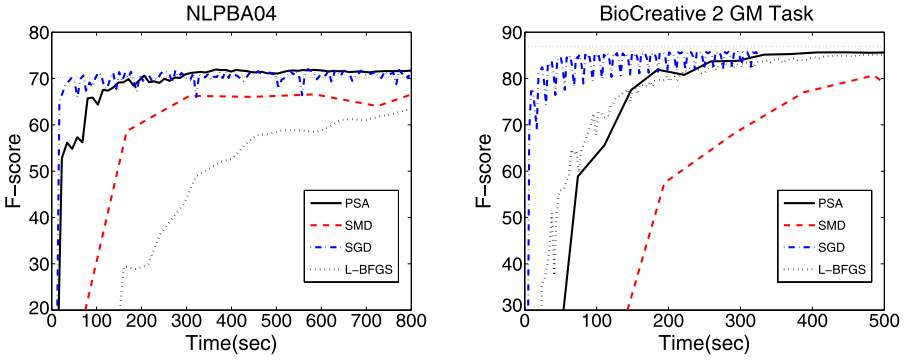
5.1.4 Guideline for parameter assignment

We provide here an empirically derived guideline for parameter assignment and discuss why this guideline is reasonable. Initially, we investigated the impact of the update frequency b for PSA. In Sect. 4.2, our analysis suggested that a too large or too small b should be avoided. We ran PSA with different values of b for the base NP task to validate this analysis. Figure 9 shows the resulting learning curves. When b is too small, the curve rises rapidly but can only converge at a poor F-score, as the case when $b = 5$. When b is between 7 to 15, the learning curves are nearly the same. We only plotted the case when $b = 10$, which is also the number we used for other experiments. When $b = 20$, the increasing of the F-score slowed down but not very obviously so we did not plot the curve here. When b is as large as 50 or 100, the tendency becomes clear. In those cases, PSA can still converge at a good F-score but the convergence is slower.

The results confirmed that a moderately large b is optimal for single-pass PSA, but it is still not precise to guide the tuning. To obtain optimal single-pass generalization performance, b should be assigned according to the expected size of training examples $|\mathsf{D}|$. We found that when $|\mathsf{D}| \gg 2000$, a value for b in the order of $\frac{0.5|\mathsf{D}|}{1000}$ is usually sufficent. This setting implies that the step size will be adjusted per $\frac{|\mathsf{D}|}{1000}$ training examples.



(a) CoNLL 2000 base NP and chunking tasks



(b) BioNLP/NLPBA and BioCreative 2

Fig. 7 Comparing F-scores achieved given the same CPU time spent by PSA, SGD, SMD and L-BFGS for four sequence labeling tasks with CRF

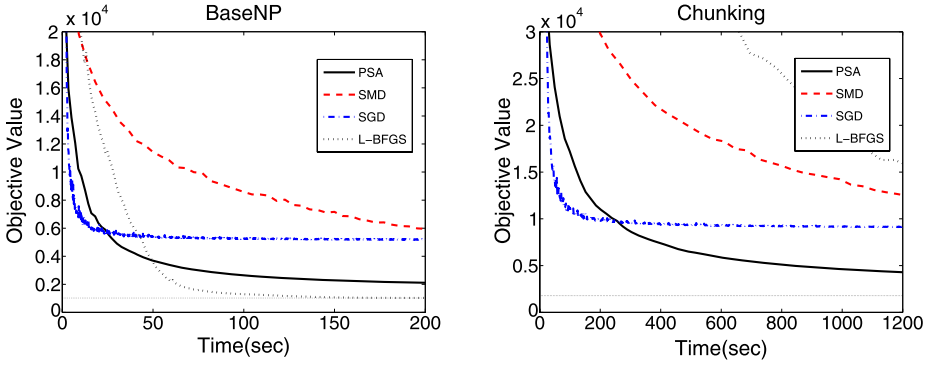
The range of b also depends on the setting of other parameters. In fact, the following settings all yield nearly identical single-pass F-score for the Base NP task:

1. $b = 10, \alpha = 0.9999, \beta = 0.99$;
2. $b = 100, \alpha = 0.999, \beta = 0.9$;
3. $b = 1, \alpha = 0.99999, \beta = 0.999$,

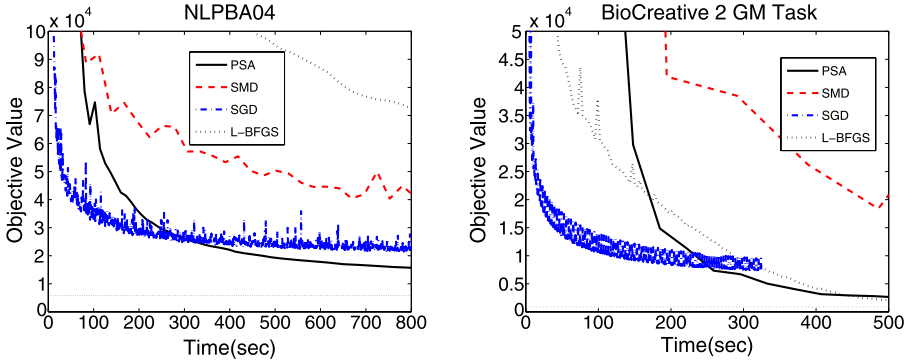
where the first setting was used in the above experiment. To see why this is the case, consider the decreasing factor v_i, which will be confined within the interval defined by (α, β). Assuming that v_i is selected by PSA at ransom uniformly, we have the mean of $v_i = 0.995$ when $(\alpha, \beta) = (0.9999, 0.99)$ and the step size η_i will be decreased by a factor of 0.995 on average in each PSA adjustment. When $b = 10$, PSA will update η_i per 20 training examples. After reading 200 training examples, PSA will decrease η_i 10 times by a combined factor of 0.9511. Similarly, we can obtain that the same factors for settings 2 and 3 are 0.95 and 0.9512, respectively, nearly identical. Consequently, we suggest select $(\alpha, \beta) = (0.9999, 0.99)$ and search for an optimal b according to the expected data size.

5.2 Linear SVM

To verify that PSA can be applied to a wide variety of models, we also evaluated PSA's single-pass performance for training linear SVM. It is straightforward to apply PSA as a



(a) CoNLL 2000 base NP and chunking tasks



(b) BioNLP/NLPBA and BioCreative 2

Fig. 8 Comparing objective (loss) values obtained given the same CPU time spent by PSA, SGD, SMD and L-BFGS for four sequence labeling tasks with CRF

Table 4 Results of bootstrap tests comparing single-pass PSA with other methods given the same CPU time spent. Cells in an italic font indicate that two methods have no significant difference, asterisked indicate that PSA achieved lower F-scores, and in a normal font indicate that PSA performed statistically significantly better

	Base NP	Chunking	NLPBA	BioCreative 2
PSA vs. SGD	*430*	*66*	4*	19*
PSA vs. SMD	1000	1000	1000	1000
PSA vs. L-BFGS	1000	1000	1000	*670*

primal optimizer for linear SVM. We used the task RCV1-C2 from (Bottou 2007) (see Table 5) and "SvmSgd," a SGD implementation for linear SVM from the same site to obtain the target empirical optimal error rate. Both PSA and SvmSgd used hinge loss. The parameter settings for PSA are $\kappa = 0.95$, $(\alpha, \beta) = (0.9999, 0.99)$, and $b = 10$, following the guideline given in Sect. 5.1.4. For $\eta^{(0)}$, SvmSgd has a function that uses a very small subset of training examples to select the value of $\eta^{(0)}$. We used this function to select $\eta_i^{(0)} = 0.1$, $\forall i$ for PSA. Finally, the mini-batch size is one. All experiments reported here for linear SVM

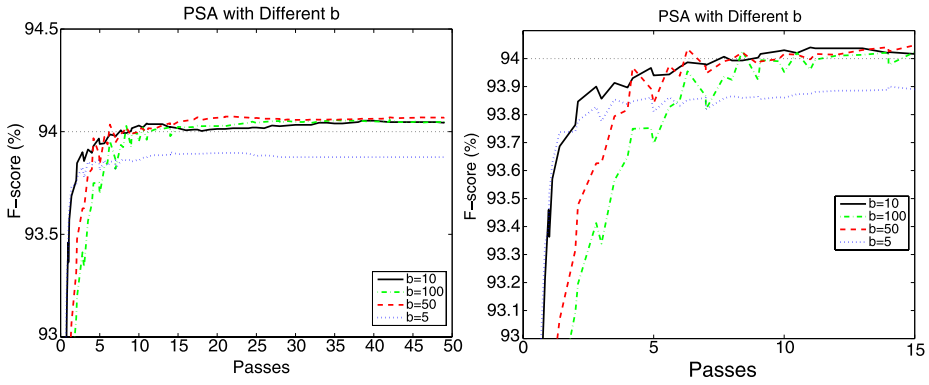


Fig. 9 Learning curves of PSA with different update frequencies for CoNLL base NP task: 50-pass view (*left*) and enlarged view (*right*)

Table 5 Tasks for the experiments on classifiers

Task	Model	Training	Test	Class	Weight	Target
RCV1-C2	SVM	781265	23149	2	47152	6.01%
MNIST-BIN	SVM	60000	10000	2	784	9.87%
MNIST	CNN	60000	10000	10	134066	0.99%

Table 6 CPU time in seconds, percentage test error rates and primal objective values by various linear SVM solvers

RCV1-C2			MNIST-BIN							
Method (pass)	time	error	Method (pass)	time	error	object	Method (pass)	time	error	object
SvmSgd (1)	0.3	6.02	SGD L1 (1)	0.56	11.90	27707	SGD L2 (1)	0.62	11.62	23068
SvmSgd (300)	87.0	6.02	PSA L1 (1)	0.69	10.48	17387	PSA L2 (1)	0.72	10.50	18683
PSA (1)	105.0	5.63					LibLinear (n/a)	3.12	9.87	n/a

were run on a Fedora 5 x86-64 Linux machine with Intel Core2 Duo E6550 CPU (2.33 GHz) and 2 GB RAM.

The experimental results are shown in Table 6. In one pass, PSA achieves a test error rate lower than the target rate. Since the data is extremely sparse and its dimensionality is very high, SGD can take full advantage of the sparse trick and complete a pass much faster than PSA. But SvmSgd converges after one pass. Its error rate never gets improved no matter how long we run it.

In a less sparse task, we used PSA to train linear SVM with the MNIST data set (LeCun and Cortes 1998) to distinguish images of hand-written odd digits from even ones (see MNIST-BIN in Table 5). To obtain target empirical optimum, we used the latest release of LibLinear (Chang and Lin 2001) (June 2, 2008). Since LibLinear used L2-norm loss, we modified our SGD and PSA programs to use the same loss functions. The L2-norm loss

function of a linear SVM given training examples $\mathsf{D} = (\mathbf{x}_i, y_i); i = 1, 2, \ldots, n$, is defined by

$$P_{\mathsf{D}}(\mathbf{w}, b) := \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))^2,$$

where $\mathbf{w}$ is the weight vector. Then the on-line loss function is:

$$P_{(\mathbf{x},y)}(\mathbf{w}, b) := \frac{1}{2n}\|\mathbf{w}\|^2 + C\max(0, 1 - y(\mathbf{w}^T\mathbf{x} + b))^2.$$

The gradient is given by:

$$\frac{\partial P}{\partial w_k} = \frac{1}{n}w_k + 2C(-yx_k + y^2w_kx_k^2 + y^2bx_k), \quad k = 1, \ldots, d,$$
$$\frac{\partial P}{\partial b} = 2C(-y + y^2\mathbf{w}^T\mathbf{x} + y^2b)$$

when $1 - y(\mathbf{w}^T\mathbf{x} + b) > 0$. Otherwise, it is:

$$\frac{\partial P}{\partial w_k} = \frac{1}{n}w_k.$$

The parameter settings for PSA for this task is the same as those for RCV1-C2 except for $\eta^{(0)}$. Again, we used SGD to select its values and assigned the values for both SGD and PSA. The selected values are 0.1 for all elements for L1(hinge)-loss and 0.01 for L2-loss.

The experimental results for MINST are shown in Table 6. In one pass, PSA achieves a test error rate close to LibLinear with the CPU time comparable to SGD. PSA achieved a much lower primal objective values than SGD. We note that for both tasks, it may take hours to complete for a conventional dual SVM solver. We did not compare with other linear SVM methods because our main purpose is to evaluate PSA's single-pass performance.

We performed the same bootstrap test as described in Sect. 5.1.3 to compare single-pass PSA and SGD given the same CPU time. The result shows that in 10,000 trials, the accuracies of PSA L2 exceeded those of SGD L2 in 10,000 times.

5.3 Convolutional neural network

Approximating Hessian is particularly challenging when the loss function is non-convex. We tested PSA in such a setting by applying PSA to train a large convolutional neural network (CNN) for the original 10-class MNIST task (see Table 5).

Let x_{ji} be the ith input of the j-th unit of the output layer in a CNN, o_j the computed output of the jth unit, and t_j the target output (answer) of the jth unit. The output o is a non-linear function:

$$o = \sigma(y) = \frac{1}{1 + e^{-y}},$$

where y is a linear function of a current weight vector $\mathbf{w}$. The loss function is the mean square error of $\mathbf{w}$ given the training data set D:

$$E(\mathbf{w}) := \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2.$$

The loss function for each training example d is:

$$E_d(\mathbf{w}) := \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2,$$

and the gradient is:

$$g_{ji} = \frac{\partial E_d}{\partial w_{ji}} = (t_j - o_j) o_j (1 - o_j) x_{ji}.$$

We tried to duplicate the implementation of LeNet described in LeCun et al. (1998b) in C++. Our implementation, referred to as "LeNet-S", is a simplified variant of LeNet-5. The differences include that the sub-sampling layers in LeNet-S picks only the upper-left value from a 2×2 area and abandons the other three, while in LeNet-5 all four values are averaged and sent to a trainable biased sigmoid node. LeNet-S used more maps (50 vs. 16) in the third layer where LeNet-5 chooses inputs from the second layer based on a user-defined template filter. In the fifth layer, LeNet-S used less nodes (120 vs. 100) due to the difference in the previous sub-sampling layer. Finally, we did not implement the Gaussian connections in the last layer.

We trained LeNet-S by plain SGD and PSA. The initial η for SGD was 0.7 and decreased by 3 percent per pass. For PSA, we used $\kappa = 0.9$, $(\alpha, \beta) = (0.99999, 0.999)$, $b = 10$, $\eta_i^{(0)} = 0.5$, $\forall i$, and the mini-batch size is one for all trials. We also adapted a trick given in LeCun et al. (1998a) which advises that step sizes in the lower layers should be larger than in the higher layer. Following their trick, the initial step sizes for the first and the third layers were 5 and $\sqrt{2.5}$ times as large as those for the other layers, respectively. The experiments were run on the same machine as for the CRF experiments.

Table 7 shows the results. To obtain the empirical optimal error rate of our LeNet-S model, we ran plain SGD with sufficient passes and obtained 0.99% error rate at convergence, slightly lower than LeNet-5's 0.95% (LeCun et al. 1998b). Single-pass performance of PSA with the layer trick is within one percentage point to the target. However, though single-pass PSA reached a mean square error lower than single-pass SGD, there is much room for minimization, as 140-pass SGD can reach a mean square error as low as 82.22. Starting from an initial weight closer to the optimum helped improving PSA's performance further. We ran SGD 100 passes with randomly selected 10 K training examples then re-started training with PSA using the rest 50 K training examples for a single pass. Though PSA did achieve a better error rate, this is infeasible because it took 4492 seconds to run SGD 100 passes. Finally, though not directly comparable, we also report the performance of TONGA given in LeRoux et al. (2008) as a reference. TONGA is a 2SGD method based on natural gradient. After 500 seconds, TONGA reaches 2% of error with 50 K training examples, but we have no information about the detailed configuration of their implementation.

We performed the same bootstrap test as described in Sect. 5.1.3 to compare CNN with and without the layer trick. The result shows that in 10,000 trials, the accuracies of CNN

Table 7 CPU time in seconds, percentage test error rates, and mean square errors by various neural network trainers

Method (pass)	time	error	mse	Method (pass)	time	error	mse
SGD (1)	266.77	2.36	2785.01	PSA w/o layer trick (1)	311.95	2.31	2389.65
SGD (140)	37336.20	0.99	82.22	PSA w/ layer trick (1)	311.00	1.97	2112.77
TONGA (n/a)	500.00	2.00	n/a	PSA re-start (1)	253.72	1.90	–

with the layer trick exceeded those of without the layer trick in 9986 times, implying that the result reported here is statistically significant.

6 Conclusions

It has been shown that given a sufficiently large training set, a single pass of 2SGD generalizes as well as the empirical optimum. Our results show that PSA provides a practical solution to accomplish near optimal performance of 2SGD as predicted theoretically for a variety of large scale models and tasks with a reasonably low cost per iteration compared to competing 2SGD methods. The benefit of 2SGD with PSA over plain SGD becomes clearer when the scale of the tasks are increasingly large. For non-convex neural network tasks, since the curvature of the error surface is so complex, it is still very challenging for an eigenvalue approximation method like PSA. Our future work is to extend PSA to kernel methods.

Acknowledgements We thank the anonymous reviewers for their helpful comments. This research was supported in part by the National Research Program in Genomic Medicine (NRPGM), National Science Council, Taiwan, under Grant No. NSC98-3112-B-001-026 (Bioinformatics Core Service), in part by National Science Council, Taiwan under Grant No. NSC97-2221-E-001-021-MY2, and in part by a cross-university collaboration fund from National Taiwan University of Science and Technology.

References

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*, 251–276.

Becker, S., & LeCun, Y. (1988). Improving the convergence of back-propagation learning with second-order methods. In *Proceedings of the 1988 connectionist models summer school* (pp. 29–37). San Mateo: Morgan Kaufman.

Benveniste, A., Metivier, M., & Priouret, P. (1990). *Adaptive algorithms and stochastic approximations*. Berlin: Springer.

Bordes, A., Bottou, L., Gallinari, P., & Weston, J. (2007). Solving multiclass support vector machines with LaRank. In *Proceedings of the 24th international conference on machine learning (ICML'07)* (pp. 89–96). New York: ACM.

Bottou, L. (2007). *The tradeoffs of large-scale learning*. Tutorial, the 21st annual conference on neural information processing systems (NIPS 2007), Vancouver, BC, Canada. http://leon.bottou.org/talks/largescale.

Bottou, L., & Bousquet, O. (2008). The tradeoffs of large scale learning. In *Advances in neural information processing systems, 20 (NIPS 2007)*. Cambridge: MIT Press.

Bottou, L., & LeCun, Y. (2004). Large scale online learning. In S. Thrun, L. Saul, & B. Schölkopf (Eds.), *Advances in neural information processing systems 16 (NIPS 2003)*. Cambridge: MIT Press.

Bottou, L., & LeCun, Y. (2005). On-line learning for very large data sets. *Applied Stochastic Models in Business and Industry*, *21*(2), 137–151.

Bottou, L., Chapelle, O., DeCoste, D., & Weston, J. (Eds.) (2007). *Large scale kernel machines*. Cambridge: MIT Press.

Burden, R. L., & Faires, D. (1988). *Numerical analysis*. PWS-KENT Pub. Co.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chen, S. F., & Rosenfeld, R. (1999). *A Gaussian prior for smoothing maximum entropy models*. Technical Report CMU-CS-99-108, School of Computer Science, Carnegie Mellon University, PA, USA, Pittsburgh.

Collins, M., Globerson, A., Koo, T., Carreras, X., & Bartlett, P. L. (2008). Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *Journal of Machine Learning Research*, *9*, 1775–1822.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, *39*(1), 1–38.

Fahlmann, S. E. (1988). *An empirical study of learning speed in back-propagation networks*. Technical Report CMU-CS-88-162, Carnegie Mellon University.

Fraley, C. (1999). On computing the largest fraction of missing information for the EM algorithm and the worst linear function for data augmentation. *Computational Statistics and Data Analysis*, *31*(1), 13–26.

Hesterberg, T. (2005). Staggered Aitken acceleration for EM. In *Proceedings of the statistical computing section of the American statistical association*, Minneapolis, Minnesota, USA.

Hsu, C.-N., Huang, H.-S., & Yang, B.-H. (2006). Global and componentwise extrapolation for accelerating data mining from large incomplete data sets with the EM algorithm. In *Proceedings of the sixth IEEE international conference on data mining (ICDM'06)* (pp. 265–274), Hong Kong, China.

Hsu, C.-N., Chang, Y.-M., Kuo, C.-J., Lin, Y.-S., Huang, H.-S., & Chuang, I.-F. (2008). Integrating high dimensional bi-directional parsing models for gene mention tagging. *Bioinformatics*, *24*(13), i286–i294. Proceedings of ISMB-2008.

Huang, H.-S., Yang, B.-H., & Hsu, C.-N. (2005). Triple-jump acceleration for the EM algorithm. In *Proceedings of the fifth IEEE international conference on data mining (ICDM'05)* (pp. 649–652).

Huang, H.-S., Yang, B.-H., Chang, Y.-M., & Hsu, C.-N. (2009). Global and componentwise extrapolations for accelerating training of Bayesian networks and conditional random fields. *Data Mining and Knowledge Discovery*, *19*(1), 58–91.

Joachims, T. (1998). Making large-scale svm learning practical. In B. Schölkoph, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods: support vector learning* (Chap. 11, pp. 169–184). Cambridge: MIT Press.

Joachims, T. (2006). Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06)* (pp. 217–226). New York: ACM.

Kudo, T. (2006). CRF++: *Yet another CRF toolkit*. Available under LGPL from the following URL: http://crfpp.sourceforge.net/.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of 18th international conference on machine learning (ICML'03)* (pp. 282–289).

LeCun, Y., & Cortes, C. (1998). *The MNIST database of handwritten digits*. http://yann.lecun.com/exdb/mnist/.

LeCun, Y., Bottou, L., Orr, G. B., & Muller, K.-R. (1998a). Efficient backprop. In G. Orr & K. Muller (Eds.), *Neural networks: tricks of the trade*. Berlin: Springer.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998b). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision* (p. 319). London: Springer.

LeRoux, N., Manzagol, P.-A., & Bengio, Y. (2008). Topmoumoute online natural gradient algorithm. In *Advances in neural information processing systems, 20 (NIPS 2007)*. Cambridge: MIT Press.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the sixth conference on natural language learning (CoNLL-2002)* (pp. 49–55).

McLachlan, G. J., & Krishnan, T. (1997). *The EM algorithm and extensions*. *Wiley series in probability and statistics*. New York: Wiley-Interscience.

Meng, X.-L., & Rubin, D. B. (1994). On the global and componentwise rates of convergence of the EM algorithm. *Linear Algebra and Its Applications*, *199*, 413–425.

Murata, N. (1998). A statistical study of on-line learning. In *On-line learning in neural networks* (pp. 63–92). Cambridge: Cambridge University Press.

Murata, N., & Amari, S.-I. (1999). Statistical analysis of learning dynamics. *Signal Processing*, *74*(1), 3–28.

Nocedal, J., & Wright, S. J. (1999). *Numerical optimization*. Berlin: Springer.

Ortega, J. M., & Rheinboldt, W. C. (1970). *Iterative solution of nonlinear equations in several variables*. San Diego: Academic Press.

Salakhutdinov, R., & Roweis, S. (2003). Adaptive overrelaxed bound optimization methods. In *Proceedings of the twentieth international conference on machine learning (ICML'03)* (pp. 664–671).

Schafer, J. L. (1997). *Analysis of incomplete multivariate data*. New York: Chapman and Hall.

Schraudolph, N. N., & Graepel, T. (2002). Conjugate directions for stochastic gradient descent. In *Proceedings of international conference on artificial neural networks (ICANN'02)* (pp. 1351–1358).

Schraudolph, N. N., Yu, J., & Günter, S. (2007). A stochastic quasi-newton method for online convex optimization. In *Proceedings of the 11th international conference on artificial intelligence and statistics (AISTATS 2007)*, San Juan, Puerto Rico.

Settles, B. (2004). Biomedical named entity recognition using conditional random fields and novel feature sets. In *Proceedings of the joint workshop on natural language processing in biomedicine and its applications (JNLPBA-2004)* (pp. 104–107).

Sha, F., & Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proceedings of human language technology, the North American chapter of the association for computational linguistics (NAACL'03)* (pp. 213–220).

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *ICML'07: Proceedings of the 24th international conference on machine learning* (pp. 807–814). New York: ACM.

Spall, J. C. (2003). *Introduction to stochastic search and optimization: estimation, simulation and control*. New York: Wiley-Interscience.

Taskar, B. (2005). *Learning structured prediction models: a large margin approach*. Ph.D. thesis, Stanford University, Stanford, CA, USA. Adviser-Daphne Koller.

Taskar, B., Chatalbashev, V., Koller, D., & Guestrin, C. (2005). Learning structured prediction models: a large margin approach. In *Proceedings of the 22nd international conference on machine learning (ICML'05)* (pp. 896–903). New York: ACM.

Traub, J. (1964). *Iterative methods for the solution of equations*. Englewood Cliffs: Prentice Hall.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Vishwanathan, S., Schraudolph, N. N., Schmidt, M. W., & Murphy, K. P. (2006a). Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the 23rd international conference on machine learning (ICML'06)*, Pittsburgh, PA, USA.

Vishwanathan, S., Schraudolph, N. N., & Smola, A. J. (2006b). Step size adaptation in reproducing kernel Hilbert space. *Journal of Machine Learning Research*, *7*, 1107–1133.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON convention record* (pp. 96–104), New York.