Journal of Machine Learning Research 10 (2009) 1737-1754 Submitted 1/09; Revised 4/09; Published 7/09

SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent

Antoine Bordes[*] ANTOINE.BORDES@LIP6.FR
LIP6, Université Pierre et Marie Curie
104, Avenue du Président Kennedy
75016 Paris, France

Léon Bottou LEONB@NEC-LABS.COM
NEC Laboratories America, Inc.
4 Independence Way
Princeton, NJ 08540, USA

Patrick Gallinari PATRICK.GALLINARI@LIP6.FR
LIP6, Université Pierre et Marie Curie
104, Avenue du Président Kennedy
75016 Paris, France

Editors: Soeren Sonnenburg, Vojtech Franc, Elad Yom-Tov and Michele Sebag

Abstract

The SGD-QN algorithm is a stochastic gradient descent algorithm that makes careful use of second-order information and splits the parameter update into independently scheduled components. Thanks to this design, SGD-QN iterates nearly as fast as a first-order stochastic gradient descent but requires less iterations to achieve the same accuracy. This algorithm won the "Wild Track" of the first PASCAL Large Scale Learning Challenge (Sonnenburg et al., 2008).

Keywords: support vector machine, stochastic gradient descent

1. Introduction

The last decades have seen a massive increase of data quantities. In various domains such as biology, networking, or information retrieval, fast classification methods able to scale on millions of training instances are needed. Real-world applications demand learning algorithms with low time and memory requirements. The first PASCAL Large Scale Learning Challenge (Sonnenburg et al., 2008) was designed to identify which machine learning techniques best address these new concerns. A generic evaluation framework and various data sets have been provided. Evaluations were carried out on the basis of various performance curves such as training time versus test error, data set size versus test error, and data set size versus training time.[1]

Our entry in this competition, named SGD-QN, is a carefully designed *Stochastic Gradient Descent* (SGD) for *linear Support Vector Machines* (SVM).

Nonlinear models could in fact reach much better generalization performance on most of the proposed data sets. Unfortunately, even in the Wild Track case, the evaluation criteria for the competition reward good scaling properties and short training durations more than they punish suboptimal test errors. Nearly all the competitors chose to implement linear models in order to avoid the

∗. Also at NEC Laboratories America, Inc.

1. This material and its documentation can be found at `http://largescale.first.fraunhofer.de/`.

additional penalty implied by nonlinearities. Although SGD-QN can work on nonlinear models,[2] we only report its performance in the context of linear SVMs.

Stochastic algorithms are known for their poor optimization performance. However, in the large scale setup, when the bottleneck is the computing time rather than the number of training examples, Bottou and Bousquet (2008) have shown that stochastic algorithms often yield the best generalization performances in spite of being worst optimizers. SGD algorithms were therefore a natural choice for the "Wild Track" of the competition which focuses on the relation between training time and test performance.

SGD algorithms have been the object of a number of recent works. Bottou (2007) and Shalev-Shwartz et al. (2007) demonstrate that the plain Stochastic Gradient Descent yields particularly effective algorithms when the input patterns are very sparse, taking less than $O(d)$ space and time per iteration to optimize a system with d parameters. It can greatly outperform sophisticated batch methods on large data sets but suffers from slow convergence rates especially on ill-conditioned problems. Various remedies have been proposed: *Stochastic Meta-Descent* (Schraudolph, 1999) heuristically determines a learning rate for each coefficient of the parameter vector. Although it can solve some ill-conditioning issues, it does not help much for linear SVMs. *Natural Gradient Descent* (Amari et al., 2000) replaces the learning rate by the inverse of the Riemannian metric tensor. This quasi-Newton stochastic method is statistically efficient but is penalized in practice by the cost of storing and manipulating the metric tensor. *Online BFGS* (oBFGS) and *Online Limited storage BFGS* (oLBFGS) (Schraudolph et al., 2007) are stochastic adaptations of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm. The limited storage version of this algorithm is a quasi-Newton stochastic method whose cost by iteration is a small multiple of the cost of a standard SGD iteration. Unfortunately this penalty is often bigger than the gains associated with the quasi-Newton update. *Online Dual Solvers* for SVMs (Bordes et al., 2005; Hsieh et al., 2008) have also shown good performance on large scale data sets. These solvers can be applied to both linear and nonlinear SVMs. In the linear case, these dual algorithms are surprising close to SGD but do not require fiddling with learning rates. Although this is often viewed as an advantage, we feel that this aspect restricts the improvement opportunities.

The contributions of this paper are twofold:

1. We conduct an analysis of different factors, ranging from algorithmic refinements to implementation details, which can affect the learning speed of SGD algorithms.

2. We present a novel algorithm, denoted SGD-QN, that carefully exploits these speedup opportunities. We empirically validate its properties by benchmarking it against state-of-the-art SGD solvers and by summarizing its results at the PASCAL Large Scale Learning Challenge (Sonnenburg et al., 2008).

The paper is organized as follows: Section 2 analyses the potential gains of quasi-Newton techniques for SGD algorithms. Sections 3 and 4 discuss the sparsity and implementation issues. Finally Section 5 presents the SGD-QN algorithm, and Section 6 reports experimental results.

2. Stochastic gradient works well in models with nonlinear parametrization. For SVMs with nonlinear kernels, we would prefer dual methods, (e.g., Bordes et al., 2005), which can exploit the sparsity of the kernel expansion.

2. Analysis

This section describes our notations and summarizes theoretical results that are relevant to the design of a fast variant of stochastic gradient algorithms.

2.1 SGD for Linear SVMs

Consider a binary classification problem with examples $z = (\mathbf{x}, y) \in \mathbb{R}^d \times \{-1,+1\}$. The linear SVM classifier is obtained by minimizing the primal cost function

$$\mathcal{P}_n(\mathbf{w}) \;=\; \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(y_i \mathbf{w}^\top \mathbf{x}_i) \;=\; \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\lambda}{2}\|\mathbf{w}\|^2 + \ell(y_i \mathbf{w}^\top \mathbf{x}_i)\right), \tag{1}$$

where the hyper-parameter $\lambda > 0$ controls the strength of the regularization term. Although typical SVMs use mildly non regular convex loss functions, we assume in this paper that the loss $\ell(s)$ is convex and twice differentiable with continuous derivatives ($\ell \in C^2[\mathbb{R}]$). This could be simply achieved by smoothing the traditional loss functions in the vicinity of their non regular points.

Each iteration of the SGD algorithm consists of drawing a random training example $(\mathbf{x}_t, y_t)$ and computing a new value of the parameter $\mathbf{w}_t$ as

$$\mathbf{w}_{t+1} \;=\; \mathbf{w}_t - \frac{1}{t+t_0}\mathbf{B}\,\mathbf{g}_t(\mathbf{w}_t) \quad \text{where} \quad \mathbf{g}_t(\mathbf{w}_t) = \lambda \mathbf{w}_t + \ell'(y_t \mathbf{w}_t^\top \mathbf{x}_t)\, y_t\, \mathbf{x}_t \tag{2}$$

where the *rescaling matrix* $\mathbf{B}$ is positive definite. Since the SVM theory provides simple bounds on the norm of the optimal parameter vector (Shalev-Shwartz et al., 2007), the positive constant t_0 is heuristically chosen to ensure that the first few updates do not produce a parameter with an implausibly large norm.

- The traditional first-order SGD algorithm, with decreasing learning rate, is obtained by setting $\mathbf{B} = \lambda^{-1}\mathbf{I}$ in the generic update (2):

$$\mathbf{w}_{t+1} \;=\; \mathbf{w}_t - \frac{1}{\lambda(t+t_0)}\,\mathbf{g}_t(\mathbf{w}_t)\,. \tag{3}$$

- The second-order SGD algorithm is obtained by setting $\mathbf{B}$ to the inverse of the Hessian Matrix $\mathbf{H} = [\,\mathcal{P}_n''(\mathbf{w}_n^*)\,]$ computed at the optimum $\mathbf{w}_n^*$ of the primal cost $\mathcal{P}_n(\mathbf{w})$:

$$\mathbf{w}_{t+1} \;=\; \mathbf{w}_t - \frac{1}{t+t_0}\,\mathbf{H}^{-1}\,\mathbf{g}_t(\mathbf{w}_t)\,. \tag{4}$$

Randomly picking examples could lead to expensive random accesses to the slow memory. In practice, one simply performs sequential passes over the randomly shuffled training set.

2.2 What Matters Are the Constant Factors

Bottou and Bousquet (2008) characterize the asymptotic learning properties of stochastic gradient algorithms in the *large scale regime*, that is, when the bottleneck is the computing time rather than the number of training examples.

The first three columns of Table 2.2 report the time for a single iteration, the number of iterations needed to reach a predefined accuracy ρ, and their product, the time needed to reach accuracy ρ.

Stochastic Gradient Algorithm	Cost of one iteration	Iterations to reach ρ	Time to reach accuracy ρ	Time to reach $\mathcal{E} \leq c\,(\mathcal{E}_{\text{app}} + \varepsilon)$
1^{st} Order SGD	$O(d)$	$\frac{\nu\kappa^2}{\rho} + o\left(\frac{1}{\rho}\right)$	$O\left(\frac{d\nu\kappa^2}{\rho}\right)$	$O\left(\frac{d\nu\kappa^2}{\varepsilon}\right)$
2^{nd} Order SGD	$O(d^2)$	$\frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right)$	$O\left(\frac{d^2\nu}{\rho}\right)$	$O\left(\frac{d^2\nu}{\varepsilon}\right)$

Table 1: Asymptotic results for stochastic gradient algorithms, reproduced from Bottou and Bousquet (2008). Compare the second last column (time to optimize) with the last column (time to reach the excess test error ε). *Legend*: n number of examples; d parameter dimension; c positive constant that appears in the generalization bounds; κ condition number of the Hessian matrix $\mathbf{H}$; $\nu = \mathbf{tr}\left(\mathbf{G}\mathbf{H}^{-1}\right)$ with $\mathbf{G}$ the Fisher matrix (see Theorem 1 for more details). The implicit proportionality coefficients in notations $O()$ and $o()$ are of course independent of these quantities.

The excess test error $\mathcal{E}$ measures how much the test error is worse than the best possible error for this problem. Bottou and Bousquet (2008) decompose the test error as the sum of three terms $\mathcal{E} = \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}}$. The *approximation error* $\mathcal{E}_{\text{app}}$ measures how closely the chosen family of functions can approximate the optimal solution, the *estimation error* $\mathcal{E}_{\text{est}}$ measures the effect of minimizing the empirical risk instead of the expected risk, the *optimization error* $\mathcal{E}_{\text{opt}}$ measures the impact of the approximate optimization on the generalization performance.

The fourth column of Table 2.2 gives the time necessary to reduce the excess test error $\mathcal{E}$ below a target that depends on $\varepsilon > 0$. This is the important metric because the test error is the measure that matters in machine learning.

Both the first-order and the second-order SGD require a time inversely proportional to ε to reach the target test error. Only the constants differ. The second-order algorithm is insensitive to the condition number κ of the Hessian matrix but suffers from a penalty proportional to the dimension d of the parameter vector. Therefore, algorithmic changes that exploit the second-order information in SGD algorithms are unlikely to yield superlinear speedups. We can at best improve the constant factors.

This property is not limited to SGD algorithms. To reach an excess error ε, the most favorable generalization bounds suggest that one needs a number of examples proportional to $1/\varepsilon$. Therefore, the time complexity of any algorithm that processes a non vanishing fraction of these examples cannot scale better than $1/\varepsilon$. In fact, Bottou and Bousquet (2008) obtain slightly worse scaling laws for typical non-stochastic gradient algorithms.

2.3 Limited Storage Approximations of Second-Order SGD

Since the second-order SGD algorithm is penalized by the high cost of performing the update (2) using a full rescaling matrix $\mathbf{B} = \mathbf{H}^{-1}$, it is tempting to consider matrices that admit a sparse representation and yet approximate the inverse Hessian well enough to reduce the negative impact of the condition number κ.

The following theorem describes how the convergence speed of the generic SGD algorithm (2) is related to the spectrum of matrix **HB**.

Theorem 1 *Let $\mathbb{E}_\sigma$ denote the expectation with respect to the random selection of the examples $(\mathbf{x}_t, y_t)$ drawn independently from the training set at each iteration. Let $\mathbf{w}_n^* = \arg\min_{\mathbf{w}} \mathcal{P}_n(\mathbf{w})$ be an optimum of the primal cost. Define the Hessian matrix $\mathbf{H} = \partial^2 \mathcal{P}_n(\mathbf{w}_n^*)/\partial \mathbf{w}^2$ and the Fisher matrix $\mathbf{G} = \mathbf{G}_t = \mathbb{E}_\sigma\left[\mathbf{g}_t'(\mathbf{w}_n^*)\,\mathbf{g}_t'(\mathbf{w}_n^*)^\top\right]$. If the eigenvalues of $\mathbf{HB}$ are in range $\lambda_{\max} \geq \lambda_{\min} > 1/2$, and if the SGD algorithm* (2) *converges to $\mathbf{w}_n^*$, the following inequality holds:*

$$\frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\max}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right) \leq \mathbb{E}_\sigma\left[\mathcal{P}_n(\mathbf{w}_t) - \mathcal{P}_n(\mathbf{w}_n^*)\right] \leq \frac{\mathbf{tr}\,(\mathbf{HBGB})}{2\lambda_{\min}-1}\,t^{-1} + \mathrm{o}\left(t^{-1}\right).$$

The proof of the theorem is provided in the appendix. Note that the theorem assumes that the generic SGD algorithm converges. Convergence in the first-order case holds under very mild assumptions (e.g., Bottou, 1998). Convergence in the generic SGD case holds because it reduces to the first-order case with a the change of variable $\mathbf{w} \to \mathbf{B}^{-\frac{1}{2}}\,\mathbf{w}$. Convergence also holds under slightly stronger assumptions when the rescaling matrix $\mathbf{B}$ changes over time (e.g., Driancourt, 1994).

The following two corollaries recover the maximal number of iterations listed in Table 2.2 with $\nu = \mathbf{tr}\left(\mathbf{GH}^{-1}\right)$ and $\kappa = \lambda^{-1}\|\mathbf{H}\|$. Corollary 2 gives a very precise equality for the second-order case because the lower bound and the upper bound of the theorem take identical values. Corollary 3 gives a much less refined bound in the first-order case.

Corollary 2 *Assume $\mathbf{B} = \mathbf{H}^{-1}$ as in the second-order SGD algorithm* (4)*. Under the assumptions of Theorem 1, we have*

$$\mathbb{E}_\sigma\left[\mathcal{P}_n(\mathbf{w}_t) - \mathcal{P}_n(\mathbf{w}_n^*)\right] \;=\; \mathbf{tr}\left(\mathbf{GH}^{-1}\right)\,t^{-1} + \mathrm{o}\left(t^{-1}\right) \;=\; \nu\,t^{-1} + \mathrm{o}\left(t^{-1}\right).$$

Corollary 3 *Assume $\mathbf{B} = \lambda^{-1}\,\mathbf{I}$ as in the first-order SGD algorithm* (3)*. Under the assumptions of Theorem 1, we have*

$$\mathbb{E}_\sigma\left[\mathcal{P}_n(\mathbf{w}_t) - \mathcal{P}_n(\mathbf{w}_n^*)\right] \;\leq\; \lambda^{-2}\,\mathbf{tr}\left(\mathbf{H}^2\mathbf{GH}^{-1}\right)\,t^{-1} + \mathrm{o}\left(t^{-1}\right) \;\leq\; \kappa^2\,\nu\,t^{-1} + \mathrm{o}\left(t^{-1}\right).$$

An often rediscovered property of second order SGD provides an useful point of reference:

Theorem 4 (Fabian, 1973; Murata and Amari, 1999; Bottou and LeCun, 2005)
Let $\mathbf{w}^* = \arg\min \frac{\lambda}{2}\|\mathbf{w}\|^2 + \mathbb{E}_{\mathbf{x},y}\left[\,\ell(y\,\mathbf{w}^\top\mathbf{x})\,\right]$. Given a sample of n independent examples $(\mathbf{x}_i, y_i)$, define $\mathbf{w}_n^* = \arg\min_{\mathbf{w}} \mathcal{P}_n(\mathbf{w})$ and compute $\mathbf{w}_n$ by applying the second-order SGD update (4) *to each of the n examples. If they converge, both $n\mathbb{E}\left[\,\|\mathbf{w}_n - \mathbf{w}^*\|^2\right]$ and $n\mathbb{E}\left[\,\|\mathbf{w}_n^* - \mathbf{w}^*\|^2\right]$ converge to a same positive constant K when n increases.*

This result means that, asymptotically and on average, the parameter $\mathbf{w}_n$ obtained after one pass of second-order SGD is as close to the infinite training set solution $\mathbf{w}^*$ as the true optimum of the primal $\mathbf{w}_n^*$. Therefore, when the training set is large enough, we can expect that a single pass of second-order SGD (n iterations of (4)) optimizes the primal accurately enough to replicate the test error of the actual SVM solution.

When we replace the full second-order rescaling matrix $\mathbf{B} = \mathbf{H}^{-1}$ by a more computationally acceptable approximation, Theorem 1 indicates that we lose a constant factor k on the required number of iterations to reach that accuracy. In other words, we can expect to replicate the SVM test error after k passes over the randomly reshuffled training set.

On the other hand, a well chosen approximation of the rescaling matrix can save a large constant factor on the computation of the generic SGD update (2). The best training times are therefore obtained by carefully trading the quality of the approximation for sparse representations.

	Frequency	Loss
Special example:	$\frac{n}{\texttt{skip}}$	$\frac{\lambda\,\texttt{skip}}{2}\|\mathbf{w}\|^2$
Examples 1 to n:	1	$\ell(y_i \mathbf{w}^\top \mathbf{x}_i)$

Table 2: The regularization term in the primal cost can be viewed as an additional training example with an arbitrarily chosen frequency and a specific loss function.

2.4 More Speedup Opportunities

We have argued that carefully designed quasi-Newton techniques can save a constant factor on the training times. There are of course many other ways to save constant factors:

- *Exploiting the sparsity of the patterns* (see Section 3) can save a constant factor in the cost of each first-order iteration. The benefits are more limited in the second-order case, because the inverse Hessian matrix is usually not sparse.

- *Implementation details* (see Section 4) such as compiler technology or parallelization on a predetermined number of processors can also reduce the learning time by constant factors.

Such opportunities are often dismissed as engineering tricks. However they should be considered on an equal footing with quasi-Newton techniques. Constant factors matter regardless of their origin. The following two sections provide a detailed discussion of sparsity and implementation.

3. Scheduling Stochastic Updates to Exploit Sparsity

First-order SGD iterations can be made substantially faster when the patterns $\mathbf{x}_t$ are sparse. The first-order SGD update has the form

$$\mathbf{w}_{t+1} \;=\; \mathbf{w}_t - \alpha_t \mathbf{w}_t - \beta_t \mathbf{x}_t \,, \tag{5}$$

where α_t and β_t are scalar coefficients. Subtracting $\beta_t \mathbf{x}_t$ from the parameter vector involves solely the nonzero coefficients of the pattern $\mathbf{x}_t$. On the other hand, subtracting $\alpha_t \mathbf{w}_t$ involves all d coefficients. A naive implementation of (5) would therefore spend most of the time processing this first term. Shalev-Shwartz et al. (2007) circumvent this problem by representing the parameter $\mathbf{w}_t$ as the product $s_t \mathbf{v}_t$ of a scalar and a vector. The update (5) can then be computed as $s_{t+1} = (1-\alpha_t) s_t$ and $\mathbf{v}_{t+1} = \mathbf{v}_t - \beta \mathbf{x}_t / s_{t+1}$ in time proportional to the number of nonzero coefficients in $\mathbf{x}_t$.

Although this simple approach works well for the first order SGD algorithm, it does not extend nicely to quasi-Newton SGD algorithms. A more general method consists of treating the regularization term in the primal cost (1) as an additional training example occurring with an arbitrarily chosen frequency with a specific loss function.

Consider examples with the frequencies and losses listed in Table 2 and write the average loss:

$$\frac{1}{\frac{n}{\texttt{skip}}+n}\left[\frac{n}{\texttt{skip}}\left(\frac{\lambda\,\texttt{skip}}{2}\|\mathbf{w}\|^2\right) \;+\; \sum_{i=1}^{n} \ell(y_i \mathbf{w}^\top \mathbf{x}_i)\right] = \frac{\texttt{skip}}{1+\texttt{skip}}\left[\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(y_i \mathbf{w}^\top \mathbf{x}_i)\right].$$

SGD	SVMSGD2
Require: $\lambda, \mathbf{w}_0, t_0, T$	**Require:** $\lambda, \mathbf{w}_0, t_0, T,$ `skip`
1: $t = 0$	1: $t = 0$, `count` $=$ `skip`
2: **while** $t \leq T$ **do**	2: **while** $t \leq T$ **do**
3: $\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{\lambda(t+t_0)}(\lambda \mathbf{w}_t + \ell'(y_t \mathbf{w}_t^\top \mathbf{x}_t) y_t \mathbf{x}_t)$	3: $\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{\lambda(t+t_0)} \ell'(y_t \mathbf{w}_t^\top \mathbf{x}_t) y_t \mathbf{x}_t$
4:	4: `count` $=$ `count`-1
5:	5: **if** `count` ≤ 0 **then**
6:	6: $\mathbf{w}_{t+1} = \mathbf{w}_{t+1} - \frac{\texttt{skip}}{t+t_0} \mathbf{w}_{t+1}$
7:	7: `count` $=$ `skip`
8:	8: **end if**
9: $t = t + 1$	9: $t = t + 1$
10: **end while**	10: **end while**
11: **return** $\mathbf{w}_T$	11: **return** $\mathbf{w}_T$

Figure 1: Detailed pseudo-codes of the SGD and SVMSGD2 algorithms.

Minimizing this loss is of course equivalent to minimizing the primal cost (1) with its regularization term. Applying the SGD algorithm to the examples defined in Table 2 separates the regularization updates, which involve the special example, from the pattern updates, which involve the real examples. The parameter `skip` regulates the relative frequencies of these updates. The SVMSGD2 algorithm (Bottou, 2007) measures the average pattern sparsity and picks a frequency that ensures that the amortized cost of the regularization update is proportional to the number of nonzero coefficients. Figure 1 compares the pseudo-codes of the naive first-order SGD and of the first-order SVMSGD2. Both algorithms handle the real examples at each iteration (line 3) but SVMSGD2 only performs a regularization update every `skip` iterations (line 6).

Assume s is the average proportion of nonzero coefficients in the patterns $\mathbf{x}_i$ and set `skip` to c/s where c is a predefined constant (we use $c = 16$ in our experiments). Each pattern update (line 3) requires sd operations. Each regularization update (line 6) requires d operations but occurs s/c times less often. The average cost per iteration is therefore proportional to $O(sd)$ instead of $O(d)$.

4. Implementation

In the optimization literature, a superior algorithm implemented with a slow scripting language usually beats careful implementations of inferior algorithms. This is because the superior algorithm minimizes the training error with a higher order convergence.

This is no longer true in the case of large scale machine learning because we care about the test error instead of the training error. As explained above, algorithm improvements do not improve the order of the test error convergence. They can simply improve constant factors and therefore *compete evenly with implementation improvements*. Time spent refining the implementation is time well spent.

- There are lots of methods for representing sparse vectors with sharply different computing requirement for sequential and random access. Our C++ implementation always uses a full

	Full	**Sparse**
Random access to a single coefficient:	$\Theta(1)$	$\Theta(s)$
In-place addition into a full vector of dimension d':	$\Theta(d)$	$\Theta(s)$
In-place addition into a sparse vector with s' nonzeros:	$\Theta(d+s')$	$\Theta(s+s')$

Table 3: Costs of various operations on a vector of dimension d with s nonzero coefficients.

vector for representing the parameter **w**, but handles the patterns **x** using either a full vector representation or a sparse representation as an ordered list of index/value pairs.

Each calculation can be achieved directly on the sparse representation or after a conversion to the full representation (see Table 3). Inappropriate choices have outrageous costs. For example, on a dense data set with 500 attributes, using sparse vectors increases the training time by 50%; on the sparse RCV1 data set (see Table 4), using a sparse vector to represent the parameter **w** increases the training time by more than 900%.

- Modern processors often sport specialized instructions to handle vectors and multiple cores. Linear algebra libraries, such as BLAS, may or may not use them in ways that suit our purposes. Compilation flags have nontrivial impacts on the learning times.

Such implementation improvements are often (but not always) orthogonal to the algorithmic improvements described above. The main issue consists of deciding how much development resources are allocated to implementation and to algorithm design. This trade-off depends on the available competencies.

5. SGD-QN: A Careful Diagonal Quasi-Newton SGD

As explained in Section 2, designing an efficient quasi-Newton SGD algorithm involves a careful trade-off between the sparsity of the scaling matrix representation **B** and the quality of its approximation of the inverse Hessian $\mathbf{H}^{-1}$. The two obvious choices are diagonal approximations (Becker and Le Cun, 1989) and low rank approximations (Schraudolph et al., 2007).

5.1 Diagonal Rescaling Matrices

Among numerous practical suggestions for running SGD algorithm in multilayer neural networks, Le Cun et al. (1998) emphatically recommend to rescale each input space feature in order to improve the condition number κ of the Hessian matrix. In the case of a linear model, such preconditioning is similar to using a constant diagonal scaling matrix.

Rescaling the input space defines transformed patterns $\mathbf{X}_t$ such that $[\mathbf{X}_t]_i = b_i [\mathbf{x}_t]_i$ where the notation $[\mathbf{v}]_i$ represents the i-th coefficient of vector **v**. This transformation does not change the classification if the parameter vectors are modified as $[\mathbf{W}_t]_i = [\mathbf{w}_t]_i / b_i$. The first-order SGD update on these modified variable is then

$$\begin{aligned} \forall i = 1 \dots d \qquad [\mathbf{W}_{t+1}]_i &= [\mathbf{W}_t]_i - \eta_t \left(\lambda [\mathbf{W}_t]_i + \ell'(y_t \mathbf{W}_t^\top \mathbf{X}_t) y_t [\mathbf{X}_t]_i, \right) \\ &= [\mathbf{W}_t]_i - \eta_t \left(\lambda [\mathbf{W}_t]_i + \ell'(y_t \mathbf{w}_t^\top \mathbf{x}_t) y_t b_i [\mathbf{x}_t]_i \right) . \end{aligned}$$

Multiplying by b_i shows how the original parameter vector $\mathbf{w}_t$ is affected:

$$\forall i = 1 \dots d \qquad [\mathbf{w}_{t+1}]_i = [\mathbf{w}_t]_i - \eta_t \left(\lambda [\mathbf{w}_t]_i + \ell'(y_t \mathbf{w}_t^\top \mathbf{x}_t) y_t b_i^2 [\mathbf{x}_t]_i \right) .$$

We observe that rescaling the input is equivalent to multiplying the gradient by a fixed diagonal matrix $\mathbf{B}$ whose elements are the squares of the coefficients b_i.

Ideally we would like to make the product $\mathbf{BH}$ spectrally close the identity matrix. Unfortunately we do not know the value of the Hessian matrix $\mathbf{H}$ at the optimum $\mathbf{w}_n^*$. Instead we could consider the current value of the Hessian $\mathbf{H}_{\mathbf{w}_t} = \mathcal{P}''(\mathbf{w}_t)$ and compute the diagonal rescaling matrix $\mathbf{B}$ that makes $\mathbf{BH}_{\mathbf{w}_t}$ closest to the identity. This computation could be very costly because it involves the full Hessian matrix. Becker and Le Cun (1989) approximate the optimal diagonal rescaling matrix by inverting the diagonal coefficients of the Hessian. The method relies on the analytical derivation of these diagonal coefficients for multilayer neural networks. This derivation does not extend to arbitrary models. It certainly does not work in the case of traditional SVMs because the hinge loss has zero curvature almost everywhere.

5.2 Low Rank Rescaling Matrices

The popular LBFGS optimization algorithm (Nocedal, 1980) maintains a low rank approximation of the inverse Hessian by storing the k most recent rank-one BFGS updates instead of the full inverse Hessian matrix. When the successive full gradients $\mathcal{P}_n'(\mathbf{w}_{t-1})$ and $\mathcal{P}_n'(\mathbf{w}_t)$ are available, standard rank one updates can be used to directly estimate the inverse Hessian matrix $\mathbf{H}^{-1}$. Using this method with stochastic gradient is tricky because the full gradients $\mathcal{P}_n'(\mathbf{w}_{t-1})$ and $\mathcal{P}_n'(\mathbf{w}_t)$ are not readily available. Instead we only have access to the stochastic estimates $\mathbf{g}_{t-1}(\mathbf{w}_{t-1})$ and $\mathbf{g}_t(\mathbf{w}_t)$ which are too noisy to compute good rescaling matrices.

The oLBFGS algorithm (Schraudolph et al., 2007) compares instead the derivatives $\mathbf{g}_{t-1}(\mathbf{w}_{t-1})$ and $\mathbf{g}_{t-1}(\mathbf{w}_t)$ for the same example $(\mathbf{x}_{t-1}, y_{t-1})$. This reduces the noise to an acceptable level at the expense of the computation of the additional gradient vector $\mathbf{g}_{t-1}(\mathbf{w}_t)$.

Compared to the first-order SGD, each iteration of the oLBFGS algorithm computes the additional quantity $\mathbf{g}_{t-1}(\mathbf{w}_t)$ and updates the list of k rank one updates. The most expensive part however remains the multiplication of the gradient $\mathbf{g}_t(\mathbf{w}_t)$ by the low-rank estimate of the inverse Hessian. With $k = 10$, each iteration of our oLBFGS implementation runs empirically 11 times slower than a first-order SGD iteration.

5.3 SGD-QN

The SGD-QN algorithm estimates a *diagonal rescaling matrix* using a technique inspired by oLBFGS. For any pair of parameters $\mathbf{w}_{t-1}$ and $\mathbf{w}_t$, a Taylor series of the gradient of the primal cost $\mathcal{P}$ provides the secant equation:

$$\mathbf{w}_t - \mathbf{w}_{t-1} \approx \mathbf{H}_{\mathbf{w}_t}^{-1} \left(\mathcal{P}_n'(\mathbf{w}_t) - \mathcal{P}_n'(\mathbf{w}_{t-1}) \right) . \tag{6}$$

We would then like to replace the inverse Hessian matrix $\mathbf{H}_{\mathbf{w}_t}^{-1}$ by a diagonal estimate $\mathbf{B}$

$$\mathbf{w}_t - \mathbf{w}_{t-1} \approx \mathbf{B} \left(\mathcal{P}_n'(\mathbf{w}_t) - \mathcal{P}_n'(\mathbf{w}_{t-1}) \right) .$$

Since we are designing a stochastic algorithm, we do not have access to the full gradient $\mathcal{P}_n'$. Following oLBFGS, we replace them by the local gradients $\mathbf{g}_{t-1}(\mathbf{w}_t)$ and $\mathbf{g}_{t-1}(\mathbf{w}_{t-1})$ and obtain

$$\mathbf{w}_t - \mathbf{w}_{t-1} \approx \mathbf{B} \left(\mathbf{g}_{t-1}(\mathbf{w}_t) - \mathbf{g}_{t-1}(\mathbf{w}_{t-1}) \right) .$$

Since we chose to use a diagonal rescaling matrix $\mathbf{B}$, we can write the term-by-term equality

$$[\mathbf{w}_t - \mathbf{w}_{t-1}]_i \approx \mathbf{B}_{ii} \, [\mathbf{g}_{t-1}(\mathbf{w}_t) - \mathbf{g}_{t-1}(\mathbf{w}_{t-1})]_i \,,$$

where the notation $[\mathbf{v}]_i$ still represents the i-th coefficient of vector $\mathbf{v}$. This leads to computing $\mathbf{B}_{ii}$ as the average of the ratio $[\mathbf{w}_t - \mathbf{w}_{t-1}]_i / [\mathbf{g}_{t-1}(\mathbf{w}_t) - \mathbf{g}_{t-1}(\mathbf{w}_{t-1})]_i$. An online estimation is easily achieved during the course of learning by performing a leaky average of these ratios,

$$\mathbf{B}_{ii} \leftarrow \mathbf{B}_{ii} + \frac{2}{r} \left(\frac{[\mathbf{w}_t - \mathbf{w}_{t-1}]_i}{[\mathbf{g}_{t-1}(\mathbf{w}_t) - \mathbf{g}_{t-1}(\mathbf{w}_{t-1})]_i} - \mathbf{B}_{ii} \right) \qquad \forall i = 1 \ldots d \,,$$

and where the integer r is incremented whenever we update the matrix $\mathbf{B}$.

The weights of the scaling matrix $\mathbf{B}$ are initialized to λ^{-1} because this corresponds to the exact setup of first-order SGD. Since the curvature of the primal cost (1) is always larger than λ, the ratio $[\mathbf{g}_{t-1}(\mathbf{w}_t) - \mathbf{g}_{t-1}(\mathbf{w}_{t-1})]_i / [\mathbf{w}_t - \mathbf{w}_{t-1}]_i$ is always larger than λ. Therefore the coefficients $\mathbf{B}_{ii}$ never exceed their initial value λ^{-1}. Basically these scaling factors slow down the convergence along some axes. The speedup does not occur because we follow the trajectory faster, but because we follow a better trajectory.

Performing the weight update (2) with a diagonal rescaling matrix $\mathbf{B}$ consists in performing term-by-term operations with a time complexity that is marginally greater than the complexity of the first-order SGD (3) update. The computation of the additional gradient vector $\mathbf{g}_{t-1}(\mathbf{w}_t)$ and the reestimation of all the coefficients $\mathbf{B}_{ii}$ essentially triples the computing time of a first-order SGD iteration with non-sparse inputs (3), and is considerably slower than a first-order SGD iteration with sparse inputs implemented as discussed in Section 3.

Fortunately this higher computational cost per iteration can be nearly avoided by scheduling the reestimation of the rescaling matrix with the same frequency as the regularization updates. Section 5.1 has shown that a diagonal rescaling matrix does little more than rescaling the input variables. Since a fixed diagonal rescaling matrix already works quite well, there is little need to update its coefficients very often.

Figure 2 compares the SVMSGD2 and SGD-QN algorithms. Whenever SVMSGD2 performs a regularization update, we set the flag `updateB` to schedule a reestimation of the rescaling coefficients during the next iteration. This is appropriate because both operations have comparable computing times. Therefore the rescaling matrix reestimation schedule can be regulated with the same `skip` parameter as the regularization updates. In practice, we observe that each SGD-QN iteration demands less than twice the time of a first-order SGD iteration.

Because SGD-QN reestimates the rescaling matrix after a pattern update, special care must be taken when the ratio $[\mathbf{w}_t - \mathbf{w}_{t-1}]_i / [\mathbf{g}_{t-1}(\mathbf{w}_t) - \mathbf{g}_{t-1}(\mathbf{w}_{t-1})]_i$ has the form $0/0$ because the corresponding input coefficient $[\mathbf{x}_{t-1}]_i$ is zero. Since the secant Equation (6) is valid for any two values of the parameter vector, one can compute the ratios with parameter vectors $\mathbf{w}_{t-1}$ and $\mathbf{w}_t + \varepsilon$ and derive the correct value by continuity when $\varepsilon \to 0$. When $[\mathbf{x}_{t-1}]_i = 0$, we can write

$$\begin{aligned}
\frac{[(\mathbf{w}_t+\boldsymbol{\varepsilon})-\mathbf{w}_{t-1}]_i}{[\mathbf{g}_{t-1}(\mathbf{w}_t+\boldsymbol{\varepsilon})-\mathbf{g}_{t-1}(\mathbf{w}_{t-1})]_i} &= \frac{[(\mathbf{w}_t+\boldsymbol{\varepsilon})-\mathbf{w}_{t-1}]_i}{\lambda[(\mathbf{w}_t+\boldsymbol{\varepsilon})-\mathbf{w}_{t-1}]_i + \left(\ell'(y_{t-1}(\mathbf{w}_t+\boldsymbol{\varepsilon})^\top \mathbf{x}_{t-1}) - \ell'(y_{t-1}\mathbf{w}_{t-1}^\top \mathbf{x}_{t-1})\right) y_{t-1} [\mathbf{x}_{t-1}]_i} \\
&= \left(\lambda + \frac{\left(\ell'(y_{t-1}(\mathbf{w}_t+\boldsymbol{\varepsilon})^\top \mathbf{x}_{t-1}) - \ell'(y_{t-1}\mathbf{w}_{t-1}^\top \mathbf{x}_{t-1})\right) y_{t-1} [\mathbf{x}_{t-1}]_i}{[(\mathbf{w}_t+\boldsymbol{\varepsilon})-\mathbf{w}_{t-1}]_i} \right)^{-1} \\
&= \left(\lambda + \frac{0}{[\boldsymbol{\varepsilon}]_i} \right)^{-1} \quad \xrightarrow{\boldsymbol{\varepsilon} \to 0} \quad \lambda^{-1} .
\end{aligned}$$

SVMSGD2

Require: λ, $\mathbf{w}_0$, t_0, T, skip
$t = 0$, count = skip
while $t \leq T$ **do**
  $\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{\lambda(t+t_0)} \ell'(y_t \mathbf{w}_t^\top \mathbf{x}_t) y_t \mathbf{x}_t$
  count = count − 1
  if count ≤ 0 **then**
    $\mathbf{w}_{t+1} = \mathbf{w}_{t+1} - \text{skip}\,(t+t_0)^{-1} \mathbf{w}_{t+1}$
    count = skip
  end if
  $t = t + 1$
end while
return $\mathbf{w}_T$

SGD-QN

Require: λ, $\mathbf{w}_0$, t_0, T, skip
$t = 0$, count = skip,
$\mathbf{B} = \lambda^{-1}\mathbf{I}$; updateB = false; $r = 2$
while $t \leq T$ **do**
  $\mathbf{w}_{t+1} = \mathbf{w}_t - (t+t_0)^{-1} \ell'(y_t \mathbf{w}_t^\top \mathbf{x}_t) y_t \mathbf{B}\,\mathbf{x}_t$
  if updateB = true **then**
    $\mathbf{p}_t = \mathbf{g}_t(\mathbf{w}_{t+1}) - \mathbf{g}_t(\mathbf{w}_t)$
    $\forall i, \mathbf{B}_{ii} = \mathbf{B}_{ii} + \frac{2}{r}\left([\mathbf{w}_{t+1} - \mathbf{w}_t]_i [\mathbf{p}_t]_i^{-1} - \mathbf{B}_{ii}\right)$
    $\forall i, \mathbf{B}_{ii} = \max(\mathbf{B}_{ii}, 10^{-2}\lambda^{-1})$
    $r = r + 1$; updateB = false
  end if
  count = count − 1
  if count ≤ 0 **then**
    $\mathbf{w}_{t+1} = \mathbf{w}_{t+1} - \text{skip}\,(t+t_0)^{-1} \lambda \mathbf{B}\,\mathbf{w}_{t+1}$
    count = skip; updateB = true
  end if
  $t = t + 1$
end while
return $\mathbf{w}_T$

Figure 2: Detailed pseudo-codes of the SVMSGD2 and SGD-QN algorithms.

Data Set	Train. Ex.	Test. Ex.	Features	s	λ	t_0	*skip*
ALPHA	100,000	50,000	500	1	10^{-5}	10^6	16
DELTA	100,000	50,000	500	1	10^{-4}	10^4	16
RCV1	781,265	23,149	47,152	0.0016	10^{-4}	10^5	9,965

Table 4: Data sets and parameters used for experiments.

6. Experiments

We demonstrate the good scaling properties of SGD-QN in two ways: we present a detailed comparison with other stochastic gradient methods, and we summarize the results obtained on the PASCAL Large Scale Challenge.

Table 4 describes the three binary classification tasks we used for comparative experiments. The Alpha and Delta tasks were defined for the PASCAL Large Scale Challenge (Sonnenburg et al., 2008). We train with the first 100,000 examples and test with the last 50,000 examples of the official training sets because the official testing sets are not available. Alpha and Delta are dense data sets with relatively severe conditioning problems. The third task is the classification of RCV1 documents belonging to class CCAT (Lewis et al., 2004). This task has become a standard benchmark for linear SVMs on sparse data. Despite its larger size, the RCV1 task is much easier than the Alpha and Delta tasks. All methods discussed in this paper perform well on RCV1.

	ALPHA	RCV1
SGD	0.13	36.8
SVMSGD2	0.10	0.20
SGD-QN	0.21	0.37

Table 5: Time (sec.) for performing one pass over the training set.

The experiments reported in Section 6.4 use the hinge loss $\ell(s) = \max(0, 1-s)$. All other experiments use the squared hinge loss $\ell(s) = \frac{1}{2}(\max(0, 1-s))^2$. In practice, there is no need to make the losses twice differentiable by smoothing their behavior near $s = 0$. Unlike most batch optimizer, stochastic algorithms do not not aim directly for nondifferentiable points, but randomly hop around them. The stochastic noise implicitly smoothes the loss.

The SGD, SVMSGD2, oLBFGS, and SGD-QN algorithms were implemented using the same C++ code base.[3] All experiments are carried out in single precision. We did not experience numerical accuracy issues, probably because of the influence of the regularization term. Our implementation of oLBFGS maintains a rank 10 rescaling matrix. Setting the oLBFGS gain schedule is rather delicate. We obtained fairly good results by replicating the gain schedule of the VieCRF package.[4] We also propose a comparison with the online dual linear SVM solver (Hsieh et al., 2008) implemented in the LibLinear package.[5] We did not reimplement this algorithm because the LibLinear implementation has proved as simple and as efficient as ours.

The t_0 parameter is determined using an automatic procedure: since the size of the training set does not affect results of Theorem 1, we simply pick a subset containing 10% of the training examples, perform one SGD-QN pass over this subset with several values for t_0, and pick the value for which the primal cost decreases the most. These values are given in Table 4.

6.1 Sparsity Tricks

Table 5 illustrates the influence of the scheduling tricks described in Section 3. The table displays the training times of SGD and SVMSGD2. The only difference between these two algorithms are the scheduling tricks. SVMSGD2 trains 180 times faster than SGD on the sparse data set RCV1. This table also demonstrates that iterations of the quasi-newton SGD-QN are not prohibitively expensive.

6.2 Quasi-Newton

Figure 3 shows how the primal cost $\mathcal{P}_n(\mathbf{w})$ of the Alpha data set evolves with the number of passes (left) and the training time (right). Compared to the first-order SVMSGD2, both the oLBFGS and SGD-QN algorithms dramatically decrease the number of passes required to achieve similar values of the primal. Even if it uses a more precise approximation of the inverse Hessian, oLBFGS does not perform better after a single pass than SGD-QN. Besides, running a single pass of oLBFGS is much slower than running multiple passes of SVMSGD2 or SGD-QN. The benefits of its second-order approximation are canceled by its greater time requirements per iteration. On the other hand,

3. Implementations and experiment scripts are available in the `libsgdqn` library on `http://www.mloss.org`.
4. This can be found at `http://www.ofai.at/~jeremy.jancsary`.
5. This can be found at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

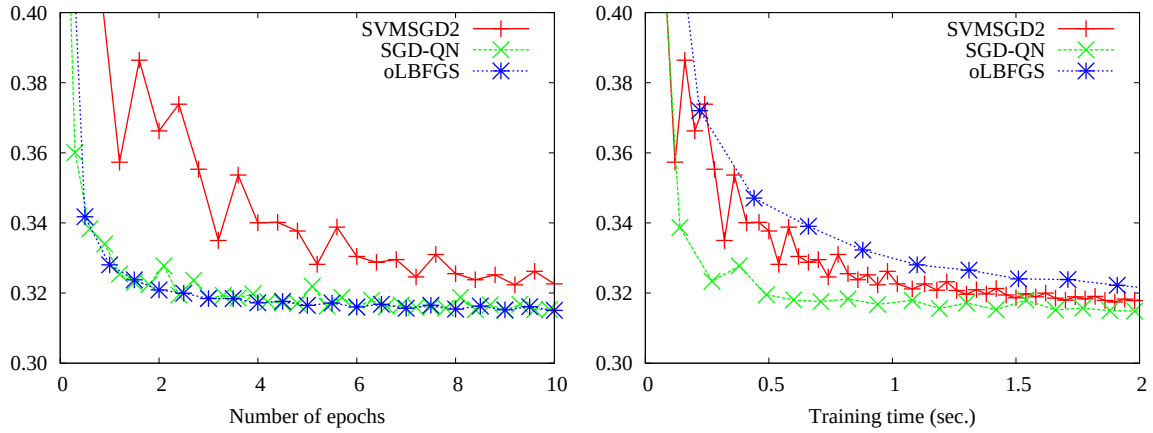


Figure 3: Primal costs according to the number of epochs (left) and the training duration (right) on the Alpha data set.

each SGD-QN iteration is only marginally slower than a SVMSGD2 iteration; the reduction of the number of iterations is sufficient to offset this cost.

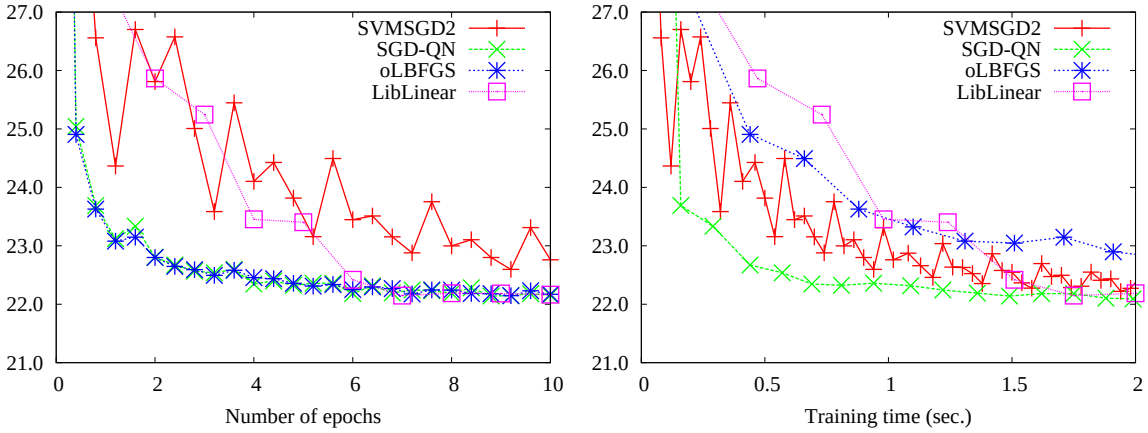
6.3 Training Speed

Figure 4 displays the test errors achieved on the Alpha, Delta and RCV1 data sets as a function of the number of passes (left) and the training time (right). These results show again that both oLBFGS and SGD-QN require less iterations than SVMSGD2 to achieve the same test error. However, oLBFGS suffers from the relatively high complexity of its update process. The SGD-QN algorithm is competitive with the dual solver LibLinear on the dense data sets Alpha and Delta; it runs significantly faster on the sparse RCV1 data set.

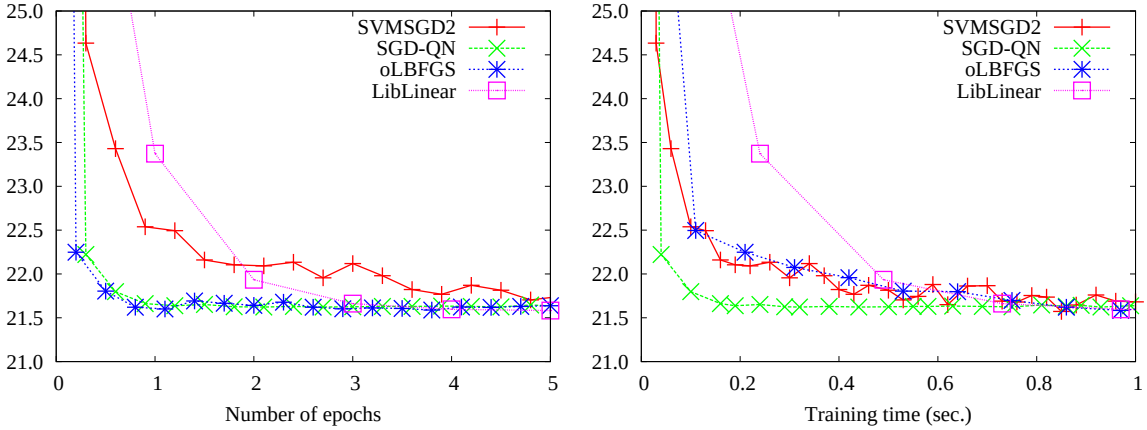
According to Theorem 4, given a large enough training set, a perfect second-order SGD algorithm would reach the batch test error after a single pass. One pass learning is attractive when we are dealing with high volume streams of examples that cannot be stored and retrieved quickly. Figure 4 (left) shows that oLBFGS is a little bit closer to that ideal than SGD-QN and could become attractive for problems where the example retrieval time is much greater than the computing time.

6.4 PASCAL Large Scale Challenge Results

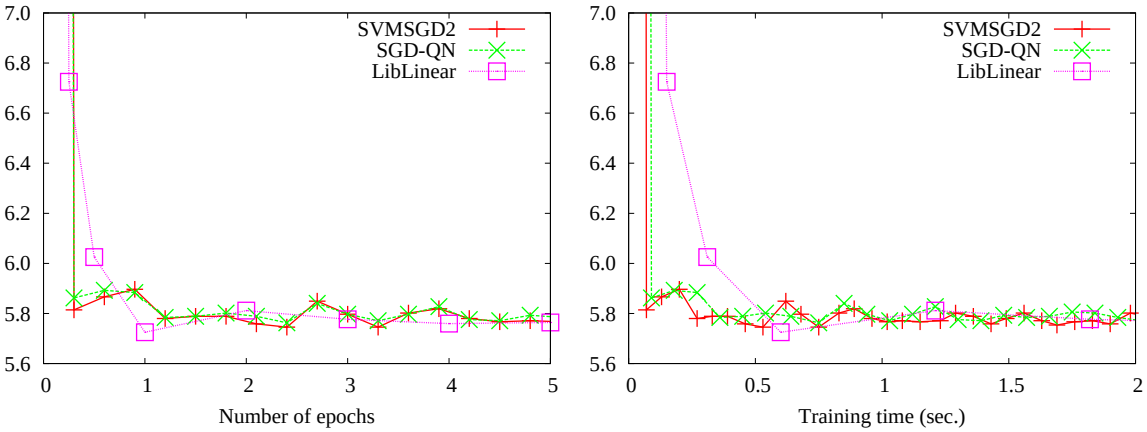
The SGD-QN algorithm has been submitted to the "Wild Track" of the PASCAL Large Scale Challenge. Wild Track contributors were free to do anything leading to more efficient and more accurate methods. Forty two methods have been submitted to this track. Table 6 shows the SGD-QN ranks determined by the organizers of the challenge according to their evaluation criteria. The SGD-QN algorithm always ranks among the top five submissions and ranks first in overall score (tie with another Newton method).



ALPHA DATA SET



DELTA DATA SET



RCV1 DATA SET

Figure 4: Test errors (in %) according to the number of epochs (left) and training duration (right).

Data Set	λ	*skip*	Passes	**Rank**
Alpha	10^{-5}	16	10	$\mathbf{1}^{st}$
Beta	10^{-4}	16	15	$\mathbf{3}^{rd}$
Gamma	10^{-3}	16	10	$\mathbf{1}^{st}$
Delta	10^{-3}	16	10	$\mathbf{1}^{st}$
Epsilon	10^{-5}	16	10	$\mathbf{5}^{th}$
Zeta	10^{-5}	16	10	$\mathbf{4}^{th}$
OCR	10^{-5}	16	10	$\mathbf{2}^{nd}$
Face	10^{-5}	16	20	$\mathbf{4}^{th}$
DNA	10^{-3}	64	10	$\mathbf{2}^{nd}$
Webspam	10^{-5}	71,066	10	$\mathbf{4}^{th}$

Table 6: Parameters and final ranks obtained by SGD-QN in the "Wild Track" of the first PASCAL Large Scale Learning Challenge. All competing algorithms were run by the organizers. (Note: the competition results were obtained with a preliminary version of SGD-QN. In particular the λ parameters listed above are different from the values used for all experiments in this paper and listed in Table 4.)

7. Conclusion

The SGD-QN algorithm strikes a good compromise for large scale application because it has low time and memory requirements per iteration and because it reaches competitive test errors after a small number of iterations. We have shown how this performance is the result of a careful design taking into account the theoretical knowledge about second-order SGD and a precise understanding of its computational requirements.

Finally, although this contribution presents SGD-QN as a solver for linear SVMs, this algorithm can be easily extended to nonlinear models for which we can analytically compute the gradients. We plan to further investigate the performance of SGD-QN in this context.

Acknowledgments

Part of this work was funded by NSF grant CCR-0325463 and by the EU Network of Excellence PASCAL2. Antoine Bordes was also supported by the French DGA.

Appendix A. Proof of Theorem 1

Define $\mathbf{v}_t = \mathbf{w}_t - \mathbf{w}_n^*$ and observe that a second-order Taylor expansion of the primal gives

$$\mathcal{P}_n(\mathbf{w}_t) - \mathcal{P}_n(\mathbf{w}_n^*) \;=\; \mathbf{v}_t^\top \mathbf{H} \mathbf{v}_t + \mathrm{o}\left(t^{-2}\right) \;=\; \mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right) + \mathrm{o}\left(t^{-2}\right).$$

Let $\mathbb{E}_{t-1}$ representing the conditional expectation over the choice of the example at iteration $t-1$ given all the choices made during the previous iterations. Since we assume that convergence takes

place, we have

$$\begin{aligned}\mathbb{E}_{t-1}\left[\mathbf{g}_{t-1}(\mathbf{w}_{t-1})\,\mathbf{g}_{t-1}(\mathbf{w}_{t-1})^\top\right] &= \mathbb{E}_{t-1}\left[\mathbf{g}_{t-1}(\mathbf{w}_n^*)\,\mathbf{g}_{t-1}(\mathbf{w}_n^*)^\top\right] + \mathrm{o}(1) \;=\; \mathbf{G} + \mathrm{o}(1)\\ \text{and } \mathbb{E}_{t-1}\left[\mathbf{g}_{t-1}(\mathbf{w}_{t-1})\right] &= \mathcal{P}_n'(\mathbf{w}_{t-1}) \;=\; \mathbf{H}\mathbf{v}_{t-1} + \mathrm{o}(\mathbf{v}_{t-1}) \;=\; \mathbf{I}_\varepsilon \mathbf{H}\mathbf{v}_{t-1}\end{aligned}$$

where notation $\mathbf{I}_\varepsilon$ is a shorthand for $\mathbf{I}+\mathrm{o}(1)$, that is, a matrix that converges to the identity. Expressing $\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top$ using the generic SGD update (2) gives

$$\begin{aligned}\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top &= \mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top - \frac{\mathbf{H}\mathbf{v}_{t-1}\,\mathbf{g}_{t-1}(\mathbf{w}_{t-1})^\top\mathbf{B}}{t+t_0} - \frac{\mathbf{H}\mathbf{B}\,\mathbf{g}_{t-1}(\mathbf{w}_{t-1})\,\mathbf{v}_{t-1}^\top}{t+t_0}\\ &\quad + \frac{\mathbf{H}\mathbf{B}\,\mathbf{g}_{t-1}(\mathbf{w}_{t-1})\,\mathbf{g}_{t-1}(\mathbf{w}_{t-1})^\top\mathbf{B}}{(t+t_0)^2}\\ \mathbb{E}_{t-1}\left[\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right] &= \mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top - \frac{\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\mathbf{H}\mathbf{I}_\varepsilon\mathbf{B}}{t+t_0} - \frac{\mathbf{H}\mathbf{B}\mathbf{I}_\varepsilon\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top}{t+t_0} + \frac{\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right)\\ \mathbb{E}_{t-1}\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] &= \mathbf{tr}\left(\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right) - \frac{2\,\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{I}_\varepsilon\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right)}{t+t_0} + \frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right)\\ \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] &= \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right)\right] - \frac{2\mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{I}_\varepsilon\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right)\right]}{t+t_0} + \frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right).\end{aligned}$$

Let $\lambda_{\max} \geq \lambda_{\min} > 1/2$ be the extreme eigenvalues of $\mathbf{HB}$. Since, for any positive matrix $\mathbf{X}$,

$$\left(\lambda_{\min} + \mathrm{o}(1)\right)\mathbf{tr}\left(\mathbf{X}\right) \;\leq\; \mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{I}_\varepsilon\mathbf{X}\right) \;\leq\; \left(\lambda_{\max} + \mathrm{o}(1)\right)\mathbf{tr}\left(\mathbf{X}\right)$$

we can bracket $\mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right]$ between the expressions

$$\left(1 - \frac{2\lambda_{\max}}{t} + \mathrm{o}\left(\frac{1}{t}\right)\right)\mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right)\right] + \frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right)$$

and

$$\left(1 - \frac{2\lambda_{\min}}{t} + \mathrm{o}\left(\frac{1}{t}\right)\right)\mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_{t-1}\mathbf{v}_{t-1}^\top\right)\right] + \frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{(t+t_0)^2} + \mathrm{o}\left(t^{-2}\right)$$

By recursively applying this bracket, we obtain

$$u_{\lambda_{\max}}(t+t_0) \;\leq\; \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] \;\leq\; u_{\lambda_{\min}}(t+t_0)$$

where the notation $u_\lambda(t)$ represents a sequence of real satisfying the recursive relation

$$u_\lambda(t) \;=\; \left(1 - \frac{2\lambda}{t} + \mathrm{o}\left(\frac{1}{t}\right)\right)u_\lambda(t-1) + \frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{t^2} + \mathrm{o}\left(\frac{1}{t^2}\right).$$

From (Bottou and LeCun, 2005, Lemma 1), $\lambda > 1/2$ implies $t\,u_\lambda(t) \longrightarrow \frac{\mathbf{tr}(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B})}{2\lambda-1}$. Then

$$\frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{2\lambda_{\max}-1}t^{-1} + \mathrm{o}\left(t^{-1}\right) \leq \mathbb{E}_\sigma\left[\mathbf{tr}\left(\mathbf{H}\mathbf{v}_t\mathbf{v}_t^\top\right)\right] \leq \frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{2\lambda_{\min}-1}t^{-1} + \mathrm{o}\left(t^{-1}\right)$$

and

$$\frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{2\lambda_{\max}-1}t^{-1} + \mathrm{o}\left(t^{-1}\right) \leq \mathbb{E}_\sigma\left[\mathcal{P}_n(\mathbf{w}_t) - \mathcal{P}_n(\mathbf{w}_n^*)\right] \leq \frac{\mathbf{tr}\left(\mathbf{H}\mathbf{B}\mathbf{G}\mathbf{B}\right)}{2\lambda_{\min}-1}t^{-1} + \mathrm{o}\left(t^{-1}\right). \qquad \blacksquare$$

References

S.-I. Amari, H. Park, and K. Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12:1409, 2000.

S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second-order methods. In *Proc. 1988 Connectionist Models Summer School*, pages 29–37. Morgan Kaufman, 1989.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *J. Machine Learning Research*, 6:1579–1619, September 2005.

L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

L. Bottou. Stochastic gradient descent on toy problems, 2007. `http://leon.bottou.org/projects/sgd`.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Adv. in Neural Information Processing Systems*, volume 20. MIT Press, 2008.

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

X. Driancourt. *Optimisation par descente de gradient stochastique de systèmes modulaires combinant réseaux de neurones et programmation dynamique*. PhD thesis, Université Paris XI, Orsay, France, 1994.

V. Fabian. Asymptotically efficient stochastic approximation; the RM case. *Annals of Statistics*, 1 (3):486–495, 1973.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proc. 25th Intl. Conf. on Machine Learning (ICML'08)*, pages 408–415. Omnipress, 2008.

Y. Le Cun, L. Bottou, G. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *J. Machine Learning Research*, 5:361–397, 2004.

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

J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *In Proc. of the 9th Intl. Conf. on Artificial Neural Networks*, pages 569–574, 1999.

N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-Newton method for online convex optimization. In *Proc. 11th Intl. Conf. on Artificial Intelligence and Statistics (AIstats)*, pages 433–440. Soc. for Artificial Intelligence and Statistics, 2007.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient solver for SVM. In *Proc. 24th Intl. Conf. on Machine Learning (ICML'07)*, pages 807–814. ACM, 2007.

S. Sonnenburg, V. Franc, E. Yom-Tov, and M. Sebag. Pascal large scale learning challenge. ICML'08 Workshop, 2008. `http://largescale.first.fraunhofer.de`.