Large-Scale Machine Learning with Stochastic Gradient Descent

Léon Bottou

NEC Labs America, Princeton NJ 08542, USA
leon@bottou.org

Abstract. During the last decade, the data sizes have grown faster than the speed of processors. In this context, the capabilities of statistical machine learning methods is limited by the computing time rather than the sample size. A more precise analysis uncovers qualitatively different tradeoffs for the case of small-scale and large-scale learning problems. The large-scale case involves the computational complexity of the underlying optimization algorithm in non-trivial ways. Unlikely optimization algorithms such as stochastic gradient descent show amazing performance for large-scale problems. In particular, second order stochastic gradient and averaged stochastic gradient are asymptotically efficient after a single pass on the training set.

Keywords: Stochastic gradient descent, Online learning, Efficiency

1 Introduction

The computational complexity of learning algorithm becomes the critical limiting factor when one envisions very large datasets. This contribution advocates stochastic gradient algorithms for large scale machine learning problems. The first section describes the stochastic gradient algorithm. The second section presents an analysis that explains why stochastic gradient algorithms are attractive when the data is abundant. The third section discusses the asymptotical efficiency of estimates obtained after a single pass over the training set. The last section presents empirical evidence.

2 Learning with gradient descent

Let us first consider a simple supervised learning setup. Each example z is a pair (x, y) composed of an arbitrary input x and a scalar output y. We consider a *loss function* $\ell(\hat{y}, y)$ that measures the cost of predicting $\hat{y}$ when the actual answer is y, and we choose a family $\mathcal{F}$ of functions $f_w(x)$ parametrized by a weight vector w. We seek the function $f \in \mathcal{F}$ that minimizes the loss $Q(z, w) = \ell(f_w(x), y)$ averaged on the examples. Although we would like to average over the unknown distribution $dP(z)$ that embodies the Laws of

Nature, we must often settle for computing the average on a sample $z_1 \dots z_n$.

$$E(f) = \int \ell(f(x), y)\, dP(z) \qquad E_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i) \tag{1}$$

The *empirical risk* $E_n(f)$ measures the training set performance. The *expected risk* $E(f)$ measures the generalization performance, that is, the expected performance on future examples. The statistical learning theory (Vapnik and Chervonenkis, 1971) justifies minimizing the empirical risk instead of the expected risk when the chosen family $\mathcal{F}$ is sufficiently restrictive.

2.1 Gradient descent

It has often been proposed (e.g., Rumelhart et al., 1986) to minimize the empirical risk $E_n(f_w)$ using *gradient descent* (GD). Each iteration updates the weights w on the basis of the gradient of $E_n(f_w)$,

$$w_{t+1} \;=\; w_t - \gamma \frac{1}{n}\sum_{i=1}^{n} \nabla_w\, Q(z_i, w_t)\,, \tag{2}$$

where γ is an adequately chosen gain. Under sufficient regularity assumptions, when the initial estimate w_0 is close enough to the optimum, and when the gain γ is sufficiently small, this algorithm achieves *linear convergence* (Dennis and Schnabel, 1983), that is, $-\log\rho \sim t$, where ρ represents the residual error.

Much better optimization algorithms can be designed by replacing the scalar gain γ by a positive definite matrix Γ_t that approaches the inverse of the Hessian of the cost at the optimum :

$$w_{t+1} \;=\; w_t - \Gamma_t \frac{1}{n}\sum_{i=1}^{n} \nabla_w\, Q(z_i, w_t)\,. \tag{3}$$

This *second order gradient descent* (2GD) is a variant of the well known Newton algorithm. Under sufficiently optimistic regularity assumptions, and provided that w_0 is sufficiently close to the optimum, second order gradient descent achieves *quadratic convergence*. When the cost is quadratic and the scaling matrix Γ is exact, the algorithm reaches the optimum after a single iteration. Otherwise, assuming sufficient smoothness, we have $-\log\log\rho \sim t$.

2.2 Stochastic gradient descent

The *stochastic gradient descent* (SGD) algorithm is a drastic simplification. Instead of computing the gradient of $E_n(f_w)$ exactly, each iteration estimates this gradient on the basis of a single randomly picked example z_t :

$$w_{t+1} \;=\; w_t - \gamma_t \nabla_w\, Q(z_t, w_t)\,. \tag{4}$$

The stochastic process $\{ w_t,\ t{=}1,\dots \}$ depends on the examples randomly picked at each iteration. It is hoped that (4) behaves like its expectation (2) despite the noise introduced by this simplified procedure.

Since the stochastic algorithm does not need to remember which examples were visited during the previous iterations, it can process examples on the fly in a deployed system. In such a situation, the stochastic gradient descent directly optimizes the expected risk, since the examples are randomly drawn from the ground truth distribution.

The convergence of stochastic gradient descent has been studied extensively in the stochastic approximation literature. Convergence results usually require decreasing gains satisfying the conditions $\sum_t \gamma_t^2 < \infty$ and $\sum_t \gamma_t = \infty$. The Robbins-Siegmund theorem (Robbins and Siegmund, 1971) provides the means to establish almost sure convergence under mild conditions (Bottou, 1998), including cases where the loss function is not everywhere differentiable.

The convergence speed of stochastic gradient descent is in fact limited by the noisy approximation of the true gradient. When the gains decrease too slowly, the variance of the parameter estimate w_t decreases equally slowly. When the gains decrease too quickly, the expectation of the parameter estimate w_t takes a very long time to approach the optimum. Under sufficient regularity conditions (e.g. Murata, 1998), the best convergence speed is achieved using gains $\gamma_t \sim t^{-1}$. The expectation of the residual error then decreases with similar speed, that is, $\mathbb{E}\,\rho \sim t^{-1}$.

The *second order stochastic gradient descent* (2SGD) multiplies the gradients by a positive definite matrix Γ_t approaching the inverse of the Hessian:

$$w_{t+1} \;=\; w_t - \gamma_t \Gamma_t \nabla_w\, Q(z_t, w_t)\,. \tag{5}$$

Unfortunately, this modification does not reduce the stochastic noise and therefore does not significantly improve the variance of w_t. Although constants are improved, the expectation of the residual error still decreases like t^{-1}, that is, $\mathbb{E}\,\rho \sim t^{-1}$, (e.g. Bordes et al., 2009, appendix).

2.3 Stochastic gradient examples

Table 1 illustrates stochastic gradient descent algorithms for a number of classic machine learning schemes. The stochastic gradient descent for the Perceptron, for the Adaline, and for k-Means match the algorithms proposed in the original papers. The SVM and the Lasso were first described with traditional optimization techniques. Both Q_{svm} and Q_{lasso} include a regularization term controlled by the hyperparameter λ. The K-means algorithm converges to a local minimum because Q_{kmeans} is nonconvex. On the other hand, the proposed update rule uses second order gains that ensure a fast convergence. The proposed Lasso algorithm represents each weight as the difference of two positive variables. Applying the stochastic gradient rule to these variables and enforcing their positivity leads to sparser solutions.

Table 1. Stochastic gradient algorithms for various learning systems.

Loss	Stochastic gradient algorithm
Adaline (Widrow and Hoff, 1960) $Q_{\text{adaline}} = \frac{1}{2}\big(y - w^\top \Phi(x)\big)^2$ $\Phi(x) \in \mathbb{R}^d,\ y = \pm 1$	$w \leftarrow w + \gamma_t \big(y_t - w^\top \Phi(x_t)\big)\, \Phi(x_t)$
Perceptron (Rosenblatt, 1957) $Q_{\text{perceptron}} = \max\{0, -y\, w^\top \Phi(x)\}$ $\Phi(x) \in \mathbb{R}^d,\ y = \pm 1$	$w \leftarrow w + \gamma_t \begin{cases} y_t\, \Phi(x_t) & \text{if } y_t\, w^\top \Phi(x_t) \le 0 \\ 0 & \text{otherwise} \end{cases}$
K-Means (MacQueen, 1967) $Q_{\text{kmeans}} = \min_k \frac{1}{2}(z - w_k)^2$ $z \in \mathbb{R}^d,\ w_1 \dots w_k \in \mathbb{R}^d$ $n_1 \dots n_k \in \mathbb{N}$, initially 0	$k^* = \arg\min_k (z_t - w_k)^2$ $n_{k^*} \leftarrow n_{k^*} + 1$ $w_{k^*} \leftarrow w_{k^*} + \frac{1}{n_{k^*}}(z_t - w_{k^*})$
SVM (Cortes and Vapnik, 1995) $Q_{\text{svm}} = \lambda w^2 + \max\{0, 1 - y\, w^\top \Phi(x)\}$ $\Phi(x) \in \mathbb{R}^d,\ y = \pm 1,\ \lambda > 0$	$w \leftarrow w - \gamma_t \begin{cases} \lambda w & \text{if } y_t\, w^\top \Phi(x_t) > 1, \\ \lambda w - y_t\, \Phi(x_t) & \text{otherwise.} \end{cases}$
Lasso (Tibshirani, 1996) $Q_{\text{lasso}} = \lambda \lvert w \rvert_1 + \frac{1}{2}\big(y - w^\top \Phi(x)\big)^2$ $w = (u_1 - v_1, \dots, u_d - v_d)$ $\Phi(x) \in \mathbb{R}^d,\ y \in \mathbb{R},\ \lambda > 0$	$u_i \leftarrow \big[u_i - \gamma_t\big(\lambda - (y_t - w^\top \Phi(x_t))\Phi_i(x_t)\big)\big]_+$ $v_i \leftarrow \big[v_i - \gamma_t\big(\lambda + (y_t - w_t^\top \Phi(x_t))\Phi_i(x_t)\big)\big]_+$ with notation $[x]_+ = \max\{0, x\}$.

3 Learning with large training sets

Let $f^* = \arg\min_f E(f)$ be the best possible prediction function. Since we seek the prediction function from a parametrized family of functions $\mathcal{F}$, let $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} E(f)$ be the best function in this family. Since we optimize the empirical risk instead of the expected risk, let $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$ be the empirical optimum. Since this optimization can be costly, let us stop the algorithm when it reaches an solution $\tilde{f}_n$ that minimizes the objective function with a predefined accuracy $E_n(\tilde{f}_n) < E_n(f_n) + \rho$.

3.1 The tradeoffs of large scale learning

The excess error $\mathcal{E} = \mathbb{E}\big[E(\tilde{f}_n) - E(f^*)\big]$ can be decomposed in three terms (Bottou and Bousquet, 2008):

$$\mathcal{E} \;=\; \mathbb{E}\big[E(f^*_{\mathcal{F}}) - E(f^*)\big] \;+\; \mathbb{E}\big[E(f_n) - E(f^*_{\mathcal{F}})\big] \;+\; \mathbb{E}\big[E(\tilde{f}_n) - E(f_n)\big]\,. \tag{6}$$

- The approximation error $\mathcal{E}_{\text{app}} = \mathbb{E}\big[E(f^*_{\mathcal{F}}) - E(f^*)\big]$ measures how closely functions in $\mathcal{F}$ can approximate the optimal solution f^*. The approximation error can be reduced by choosing a larger family of functions.
- The estimation error $\mathcal{E}_{\text{est}} = \mathbb{E}\big[E(f_n) - E(f^*_{\mathcal{F}})\big]$ measures the effect of minimizing the empirical risk $E_n(f)$ instead of the expected risk $E(f)$.

The estimation error can be reduced by choosing a smaller family of functions or by increasing the size of the training set.

- The optimization error $\mathcal{E}_{\mathrm{opt}} = E(\tilde{f}_n) - E(f_n)$ measures the impact of the approximate optimization on the expected risk. The optimization error can be reduced by running the optimizer longer. The additional computing time depends of course on the family of function and on the size of the training set.

Given constraints on the maximal computation time $T_{\max}$ and the maximal training set size $n_{\max}$, this decomposition outlines a tradeoff involving the size of the family of functions $\mathcal{F}$, the optimization accuracy ρ, and the number of examples n effectively processed by the optimization algorithm.

$$\min_{\mathcal{F},\rho,n} \quad \mathcal{E} = \mathcal{E}_{\mathrm{app}} + \mathcal{E}_{\mathrm{est}} + \mathcal{E}_{\mathrm{opt}} \quad \text{subject to } \begin{cases} n \leq n_{\max} \\ T(\mathcal{F},\rho,n) \leq T_{\max} \end{cases} \tag{7}$$

Two cases should be distinguished:

- *Small-scale learning problems* are first constrained by the maximal number of examples. Since the computing time is not an issue, we can reduce the optimization error $\mathcal{E}_{\mathrm{opt}}$ to insignificant levels by choosing ρ arbitrarily small, and we can minimize the estimation error by chosing $n = n_{\max}$. We then recover the approximation-estimation tradeoff that has been widely studied in statistics and in learning theory.
- *Large-scale learning problems* are first constrained by the maximal computing time. Approximate optimization can achieve better expected risk because more training examples can be processed during the allowed time. The specifics depend on the computational properties of the chosen optimization algorithm.

3.2 Asymptotic analysis

Solving (7) in the asymptotic regime amounts to ensuring that the terms of the decomposition (6) decrease at similar rates. Since the asymptotic convergence rate of the excess error (6) is the convergence rate of its slowest term, the computational effort required to make a term decrease faster would be wasted.

For simplicity, we assume in this section that the Vapnik-Chervonenkis dimensions of the families of functions $\mathcal{F}$ are bounded by a common constant. We also assume that the optimization algorithms satisfy all the assumptions required to achieve the convergence rates discussed in section 2. Similar analyses can be carried out for specific algorithms under weaker assumptions (e.g. Shalev-Shwartz and Srebro, 2008).

A simple application of the uniform convergence results of (Vapnik and Chervonenkis, 1971) gives then the upper bound

$$\mathcal{E} \;=\; \mathcal{E}_{\mathrm{app}} + \mathcal{E}_{\mathrm{est}} + \mathcal{E}_{\mathrm{opt}} \;=\; \mathcal{E}_{\mathrm{app}} \;+\; \mathcal{O}\left(\sqrt{\frac{\log n}{n}} + \rho \right).$$

Table 2. Asymptotic equivalents for various optimization algorithms: gradient descent (GD, eq. 2), second order gradient descent (2GD, eq. 3), stochastic gradient descent (SGD, eq. 4), and second order stochastic gradient descent (2SGD, eq. 5). Although they are the worst optimization algorithms, SGD and 2SGD achieve the fastest convergence speed on the expected risk. They differ only by constant factors not shown in this table, such as condition numbers and weight vector dimension.

	GD	**2GD**	**SGD**	**2SGD**
Time per iteration :	n	n	1	1
Iterations to accuracy ρ :	$\log \frac{1}{\rho}$	$\log\log \frac{1}{\rho}$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
Time to accuracy ρ :	$n \log \frac{1}{\rho}$	$n \log\log \frac{1}{\rho}$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
Time to excess error $\mathcal{E}$:	$\frac{1}{\mathcal{E}^{1/\alpha}} \log^2 \frac{1}{\mathcal{E}}$	$\frac{1}{\mathcal{E}^{1/\alpha}} \log \frac{1}{\mathcal{E}} \log\log \frac{1}{\mathcal{E}}$	$\frac{1}{\mathcal{E}}$	$\frac{1}{\mathcal{E}}$

Unfortunately the convergence rate of this bound is too pessimistic. Faster convergence occurs when the loss function has strong convexity properties (Lee et al., 2006) or when the data distribution satisfies certain assumptions (Tsybakov, 2004). The equivalence

$$\mathcal{E} \;=\; \mathcal{E}_{\mathrm{app}} + \mathcal{E}_{\mathrm{est}} + \mathcal{E}_{\mathrm{opt}} \;\sim\; \mathcal{E}_{\mathrm{app}} + \left(\frac{\log n}{n}\right)^{\alpha} + \rho\,, \quad \text{for some } \alpha \in \left[\tfrac{1}{2}, 1\right], \tag{8}$$

provides a more realistic view of the asymptotic behavior of the excess error (e.g. Massart, 2000, Bousquet, 2002). Since the three component of the excess error should decrease at the same rate, the solution of the tradeoff problem (7) must then obey the multiple asymptotic equivalences

$$\mathcal{E} \;\sim\; \mathcal{E}_{\mathrm{app}} \;\sim\; \mathcal{E}_{\mathrm{est}} \;\sim\; \mathcal{E}_{\mathrm{opt}} \;\sim\; \left(\frac{\log n}{n}\right)^{\alpha} \;\sim\; \rho\,. \tag{9}$$

Table 2 summarizes the asymptotic behavior of the four gradient algorithm described in section 2. The first three rows list the computational cost of each iteration, the number of iterations required to reach an optimization accuracy ρ, and the corresponding computational cost. The last row provides a more interesting measure for large scale machine learning purposes. Assuming we operate at the optimum of the approximation-estimation-optimization tradeoff (7), this line indicates the computational cost necessary to reach a predefined value of the excess error, and therefore of the expected risk. This is computed by applying the equivalences (9) to eliminate n and ρ from the third row results.

Although the stochastic gradient algorithms, SGD and 2SGD, are clearly the worst optimization algorithms (third row), they need less time than the other algorithms to reach a predefined expected risk (fourth row). Therefore, in the large scale setup, that is, when the limiting factor is the computing time rather than the number of examples, the stochastic learning algorithms performs asymptotically better !

4 Efficient learning

Let us add an additional example z_t to a training set $z_1 \dots z_{t-1}$. Since the new empirical risk $E_t(f)$ remains close to $E_{t-1}(f)$, the empirical minimum $w^*_{t+1} = \arg\min_w E_t(f_w)$ remains close to $w^*_t = \arg\min_w E_{t-1}(f_w)$. With sufficient regularity assumptions, a first order calculation gives the result

$$w^*_{t+1} \;=\; w^*_t - t^{-1}\,\Psi_t\,\nabla_w\,Q(z_t, w^*_t) + \mathcal{O}\big(t^{-2}\big)\,, \tag{10}$$

where Ψ_t is the inverse of the Hessian of $E_t(f_w)$ in w^*_t. The similarity between this expression and the second order stochastic gradient descent rule (5) has deep consequences. Let w_t be the sequence of weights obtained by performing a *single second order stochastic gradient pass* on the randomly shuffled training set. With adequate regularity and convexity assumptions, we can prove (e.g. Bottou and LeCun, 2004)

$$\lim_{t\to\infty}\; t\,\big(E(f_{w_t}) - E(f^*_{\mathcal{F}})\big) \;=\; \lim_{t\to\infty}\; t\,\big(E(f_{w^*_t}) - E(f^*_{\mathcal{F}})\big) \;=\; \mathcal{I} \;>\; 0\,. \tag{11}$$

Therefore, a single pass of second order stochastic gradient provides a prediction function f_{w_t} that approaches the optimum $f^*_{\mathcal{F}}$ as efficiently as the empirical optimum $f_{w^*_t}$. In particular, when the loss function is the log likelihood, the empirical optimum is the asymptotically efficient maximum likelihood estimate, and the second order stochastic gradient estimate is also asymptotically efficient.

Unfortunately, second order stochastic gradient descent is computationally costly because each iteration (5) performs a computation that involves the large dense matrix Γ_t. Two approaches can work around this problem.

- Computationally efficient approximations of the inverse Hessian trade asymptotic optimality for computation speed. For instance, the SGDQN algorithm (Bordes et al., 2009) achieves interesting speeds using a diagonal approximation.
- The *averaged stochastic gradient descent* (ASGD) algorithm (Polyak and Juditsky, 1992) performs the normal stochastic gradient update (4) and recursively computes the average $\bar{w}_t = \frac{1}{t}\sum_{i=1}^{t} w_t$:

  $$w_{t+1} \;=\; w_t - \gamma_t \nabla_w\,Q(z_t, w_t)\,, \quad \bar{w}_{t+1} \;=\; \frac{t}{t+1}\bar{w}_t + \frac{1}{t+1}w_{t+1}\,. \tag{12}$$

  When the gains γ_t decrease slower than t^{-1}, the $\bar{w}_t$ converges with the optimal asymptotic speed (11). Reaching this asymptotic regime can take a very long time in practice. A smart selection of the gains γ_t helps achieving the promised performance (Xu, 2010).

Algorithm	Time	Test Error
Hinge loss SVM, $\lambda = 10^{-4}$.		
SVMLight	23,642 s.	6.02 %
SVMPerf	66 s.	6.03 %
SGD	**1.4** s.	6.02 %
Log loss SVM, $\lambda = 10^{-5}$.		
TRON (`-e0.01`)	30 s.	5.68 %
TRON (`-e0.001`)	44 s.	5.70 %
SGD	**2.3** s.	5.66 %

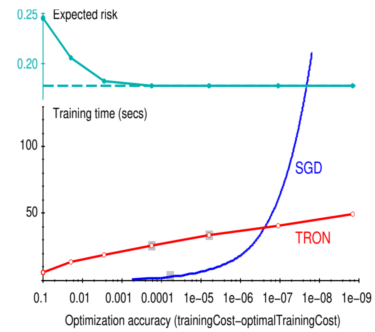

Fig. 1. Results achieved with a linear SVM on the RCV1 task. The lower half of the plot shows the time required by SGD and TRON to reach a predefined accuracy ρ on the log loss task. The upper half shows that the expected risk stops improving long before the superlinear TRON algorithm overcomes SGD.

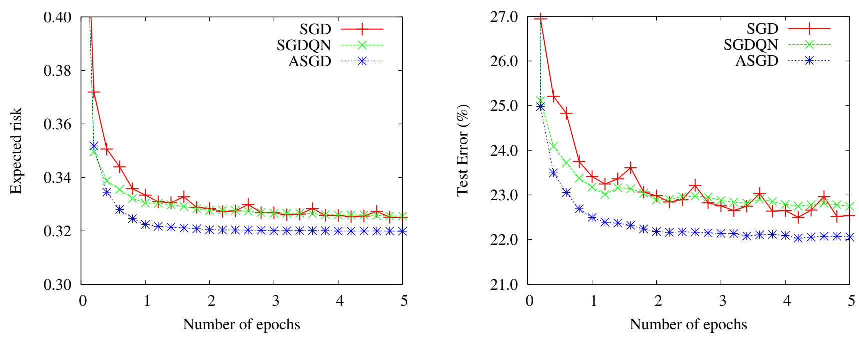

Fig. 2. Comparaison of the test set performance of SGD, SGDQN, and ASGD for a linear squared hinge SVM trained on the ALPHA task of the 2008 Pascal Large Scale Learning Challenge. ASGD nearly reaches the optimal expected risk after a single pass.

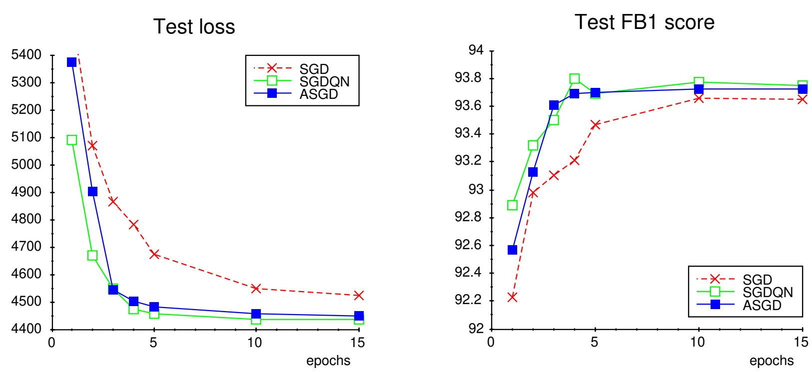

Fig. 3. Comparison of the test set performance of SGD, SGDQN, and ASGD on a CRF trained on the CONLL Chunking task. On this task, SGDQN appears more attractive because ASGD does not reach its asymptotic performance.

5 Experiments

This section briefly reports experimental results illustrating the actual performance of stochastic gradient algorithms on a variety of linear systems. We use gains $\gamma_t = \gamma_0(1 + \lambda\gamma_0 t)^{-1}$ for SGD and, following (Xu, 2010), $\gamma_t = \gamma_0(1 + \lambda\gamma_0 t)^{-0.75}$ for ASGD. The initial gains γ_0 were set manually by observing the performance of each algorithm running on a subset of the training examples.

Figure 1 reports results achieved using SGD for a linear SVM trained for the recognition of the CCAT category in the RCV1 dataset (Lewis et al., 2004) using both the hinge loss ($Q_{\rm svm}$ in table 1), and the log loss, ($Q_{\rm logsvm} = \lambda w^2 + \log(1 + \exp(-y\, w^\top \Phi(x)))$). The training set contains 781,265 documents represented by 47,152 relatively sparse TF/IDF features. SGD runs considerably faster than either the standard SVM solvers SVMLight and SVMPerf (Joachims, 2006) or the superlinear optimization algorithm TRON (Lin et al., 2007).

Figure 2 reports results achieved using SGD, SGDQN, and ASGD for a linear SVM trained on the ALPHA task of the 2008 Pascal Large Scale Learning Challenge (see Bordes et al., 2009) using the squared hinge loss ($Q_{\rm sqsvm} = \lambda w^2 + \max\{0, 1 - y\, w^\top \Phi(x)\}^2$). The training set contains 100,000 patterns represented by 500 centered and normalized variables. Performances measured on a separate testing set are plotted against the number of passes over the training set. ASGD achieves near optimal results after one pass.

Figure 3 reports results achieved using SGD, SGDQN, and ASGD for a CRF (Lafferty et al., 2001) trained on the CONLL 2000 Chunking task (Tjong Kim Sang and Buchholz, 2000). The training set contains 8936 sentences for a 1.68×10^6 dimensional parameter space. Performances measured on a separate testing set are plotted against the number of passes over the training set. SGDQN appears more attractive because ASGD does not reach its asymptotic performance. All three algorithms reach the best test set performance in a couple minutes. The standard CRF L-BFGS optimizer takes 72 minutes to compute an equivalent solution.

References

BORDES. A., BOTTOU, L., and GALLINARI, P. (2009): SGD-QN: Careful Quasi-Newton Stochastic Gradient Descent. *Journal of Machine Learning Research, 10:1737-1754.* With Erratum (to appear).

BOTTOU, L. and BOUSQUET, O. (2008): The Tradeoffs of Large Scale Learning, In *Advances in Neural Information Processing Systems*, vol.20, 161-168.

BOTTOU, L. and LECUN, Y. (2004): On-line Learning for Very Large Datasets. *Applied Stochastic Models in Business and Industry, 21(2):137-151*

BOUSQUET, O. (2002): Concentration Inequalities and Empirical Processes Theory Applied to the Analysis of Learning Algorithms. Thèse de doctorat, Ecole Polytechnique, Palaiseau, France.

CORTES, C. and VAPNIK, V. N. (1995): Support Vector Networks, *Machine Learning, 20:273-297*.

DENNIS, J. E., Jr., and SCHNABEL, R. B. (1983): *Numerical Methods For Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall

JOACHIMS, T. (2006): Training Linear SVMs in Linear Time. In *Proceedings of the 12th ACM SIGKDD*, ACM Press.

LAFFERTY, J. D., MCCALLUM, A., and PEREIRA, F. (2001): Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of ICML 2001*, 282-289, Morgan Kaufman.

LEE, W. S., BARTLETT, P. L., and WILLIAMSON, R. C. (1998): The Importance of Convexity in Learning with Squared Loss. *IEEE Transactions on Information Theory, 44(5):1974-1980*.

LEWIS, D. D., YANG, Y., ROSE, T. G., and LI, F. (2004): RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research, 5:361-397*.

LIN, C. J., WENG, R. C., and KEERTHI, S. S. (2007): Trust region Newton methods for large-scale logistic regression. In *Proceedings of ICML 2007*, 561-568, ACM Press.

MACQUEEN, J. (1967): Some Methods for Classification and Analysis of Multivariate Observations. In *Fifth Berkeley Symposium on Mathematics, Statistics, and Probabilities*, vol.1, 281-297, University of California Press.

MASSART, P. (2000): Some applications of concentration inequalities to Statistics, *Annales de la Faculté des Sciences de Toulouse, series 6,9,(2):245-303*.

MURATA, N. (1998): A Statistical Study of On-line Learning. In *Online Learning and Neural Networks*, Cambridge University Press.

POLYAK, B. T. and JUDITSKY, A. B. (1992): Acceleration of stochastic approximation by averaging. *SIAM J. Control and Optimization, 30(4):838-855*.

ROSENBLATT, F. (1957): The Perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Project PARA, Cornell Aeronautical Lab.

RUMELHART, D. E., HINTON, G. E., and WILLIAMS, R. J. (1986): Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, vol.I, 318-362, Bradford Books.

SHALEV-SHWARTZ, S. and SREBRO, N. (2008): SVM optimization: inverse dependence on training set size. In *Proceedings of the ICML 2008*, 928-935, ACM.

TIBSHIRANI, R. (1996): Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B, 58(1):267-288*.

TJONG KIM SANG E. F., and BUCHHOLZ, S. (2000): Introduction to the CoNLL-2000 Shared Task: Chunking. In *Proceedings of CoNLL-2000*, 127-132.

TSYBAKOV, A. B. (2004): Optimal aggregation of classifiers in statistical learning, *Annals of Statististics, 32(1)*.

VAPNIK, V. N. and CHERVONENKIS, A. YA. (1971): On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and its Applications, 16(2):264-280*.

WIDROW, B. and HOFF, M. E. (1960): Adaptive switching circuits. *IRE WESCON Conv. Record, Part 4., 96-104*.

XU, W. (2010): Towards Optimal One Pass Large Scale Learning with Averaged Stochastic Gradient Descent. *Journal of Machine Learning Research (to appear)*.