Stochastic Gradient Descent Tricks

Léon Bottou

Microsoft Research, Redmond, WA
leon@bottou.org
http://leon.bottou.org

Abstract. Chapter 1 strongly advocates the *stochastic back-propagation* method to train neural networks. This is in fact an instance of a more general technique called *stochastic gradient descent* (SGD). This chapter provides background material, explains why SGD is a good learning algorithm when the training set is large, and provides useful recommendations.

1 Introduction

Chapter 1 strongly advocates the *stochastic back-propagation* method to train neural networks. This is in fact an instance of a more general technique called *stochastic gradient descent* (SGD). This chapter provides background material, explains why SGD is a good learning algorithm when the training set is large, and provides useful recommendations.

2 What is Stochastic 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 $z_1 \ldots z_n$.

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

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 [25] 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., [18]) 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 learning rate. Under sufficient regularity assumptions, when the initial estimate w_0 is close enough to the optimum, and when the learning rate γ is sufficiently small, this algorithm achieves *linear convergence* [6], that is, $-\log\rho \sim t$, where ρ represents the residual error.[1]

Much better optimization algorithms can be designed by replacing the scalar learning rate 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.

[1] For mostly historical reasons, *linear convergence* means that the residual error asymptotically decreases exponentially, and *quadratic convergence* denotes an even faster asymptotic convergence. Both convergence rates are considerably faster than the SGD convergence rates discussed in section 2.3.

Table 1. Stochastic gradient algorithms for various learning systems.

Loss	Stochastic gradient algorithm
Adaline [26] $Q_{\mathrm{adaline}} = \frac{1}{2}\big(y - w^\top \Phi(x)\big)^2$ Features $\Phi(x) \in \mathbb{R}^d$, Classes $y = \pm 1$	$w \leftarrow w + \gamma_t \big(y_t - w^\top \Phi(x_t)\big)\, \Phi(x_t)$
Perceptron [17] $Q_{\mathrm{perceptron}} = \max\{0, -y\, w^\top \Phi(x)\}$ Features $\Phi(x) \in \mathbb{R}^d$, Classes $y = \pm 1$	$w \leftarrow w + \gamma_t \begin{cases} y_t\, \Phi(x_t) & \text{if } y_t\, w^\top \Phi(x_t) \le 0 \\ 0 & \text{otherwise} \end{cases}$
K-Means [12] $Q_{\mathrm{kmeans}} = \min\limits_k \frac{1}{2}(z - w_k)^2$ Data $z \in \mathbb{R}^d$ Centroids $w_1 \dots w_k \in \mathbb{R}^d$ Counts $n_1 \dots n_k \in \mathbb{N}$, initially 0	$k^* = \arg\min_k (z_t - w_k)^2$ $n_{k^*} \leftarrow n_{k^*} + 1$ $w_{k^*} \leftarrow w_{k^*} + \frac{1}{n_{k^*}}(z_t - w_{k^*})$ (counts provide optimal learning rates!)
SVM [5] $Q_{\mathrm{svm}} = \lambda w^2 + \max\{0, 1 - y\, w^\top \Phi(x)\}$ Features $\Phi(x) \in \mathbb{R}^d$, Classes $y = \pm 1$ Hyperparameter $\lambda > 0$	$w \leftarrow w - \gamma_t \begin{cases} \lambda w & \text{if } y_t\, w^\top \Phi(x_t) > 1, \\ \lambda w - y_t\, \Phi(x_t) & \text{otherwise.} \end{cases}$
Lasso [23] $Q_{\mathrm{lasso}} = \lambda \lvert w \rvert_1 + \frac{1}{2}\big(y - w^\top \Phi(x)\big)^2$ $w = (u_1 - v_1, \dots, u_d - v_d)$ Features $\Phi(x) \in \mathbb{R}^d$, Classes $y = \pm 1$ Hyperparameter $\lambda > 0$	$u_i \leftarrow \big[u_i - \gamma_t\big(\lambda - (y_t - w^\top \Phi(x_t))\Phi_i(x_t)\big)\big]_+$ $v_i \leftarrow \big[v_i - \gamma_t\big(\lambda + (y_t - w^\top \Phi(x_t))\Phi_i(x_t)\big)\big]_+$ with notation $[x]_+ = \max\{0, x\}$.

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 hyper-parameter 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 learning rates 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.

2.3 The Convergence of Stochastic Gradient Descent

The convergence of stochastic gradient descent has been studied extensively in the stochastic approximation literature. Convergence results usually require decreasing learning rates satisfying the conditions $\sum_t \gamma_t^2 < \infty$ and $\sum_t \gamma_t = \infty$. The Robbins-Siegmund theorem [16] provides the means to establish almost sure convergence under surprisingly mild conditions [3], including cases where the loss function is non smooth.

The convergence speed of stochastic gradient descent is in fact limited by the noisy approximation of the true gradient. When the learning rates decrease too slowly, the variance of the parameter estimate w_t decreases equally slowly. When the learning rates decrease too quickly, the expectation of the parameter estimate w_t takes a very long time to approach the optimum.

– When the Hessian matrix of the cost function at the optimum is strictly positive definite, the best convergence speed is achieved using learning rates $\gamma_t \sim t^{-1}$ (e.g. [14]). The expectation of the residual error then decreases with similar speed, that is, $\mathbb{E}(\rho) \sim t^{-1}$. These theoretical convergence rates are frequently observed in practice.
– When we relax these regularity assumptions, the theory suggests slower asymptotic convergence rates, typically like $\mathbb{E}(\rho) \sim t^{-1/2}$ (e.g., [28]). In practice, the convergence only slows down during the final stage of the optimization process. This may not matter in practice because one often stops the optimization before reaching this stage (see section 3.1.)

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, $\mathbb{E}(\rho) \sim t^{-1}$ at best, (e.g. [1], appendix).

Therefore, as an optimization algorithm, stochastic gradient descent is asymptotically *much slower* than a typical batch algorithm. However, this is not the whole story. . .

3 When to use Stochastic Gradient Descent?

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. The analysis presented in this section shows that stochastic gradient descent performs very well in this context.

Use stochastic gradient descent
when training time is the bottleneck.

3.1 The trade-offs of large scale learning

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 a solution $\tilde{f}_n$ that minimizes the objective function with a predefined accuracy $E_n(\tilde{f}_n) < E_n(f_n) + \rho$. The excess error $\mathcal{E} = \mathbb{E}\big[E(\tilde{f}_n) - E(f^*)\big]$ can then be decomposed in three terms [2] :

$$\mathcal{E} \;=\; \underbrace{\mathbb{E}\big[E(f^*_{\mathcal{F}}) - E(f^*)\big]}_{\mathcal{E}_{\mathrm{app}}} \;+\; \underbrace{\mathbb{E}\big[E(f_n) - E(f^*_{\mathcal{F}})\big]}_{\mathcal{E}_{\mathrm{est}}} \;+\; \underbrace{\mathbb{E}\big[E(\tilde{f}_n) - E(f_n)\big]}_{\mathcal{E}_{\mathrm{opt}}}\,. \qquad (6)$$

- The approximation error $\mathcal{E}_{\mathrm{app}} = \mathbb{E}\big[E(f^*_{\mathcal{F}}) - E(f^*)\big]$ 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 $\mathcal{E}_{\mathrm{est}} = \mathbb{E}\big[E(f_n) - E(f^*_{\mathcal{F}})\big]$ 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 $\mathcal{E}_{\mathrm{opt}} = \mathbb{E}\big[E(\tilde{f}_n) - E(f_n)\big]$ 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 trade-off 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.

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

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 $\mathcal{E}_{\mathrm{est}}$ by choosing $n = n_{\mathrm{max}}$. We then recover the approximation-estimation trade-off that has been widely studied in statistics and in learning theory.
- *Large-scale learning problems* are constrained by the maximal computing time, usually because the supply of training examples is very large. 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.

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 :	$\log \frac{1}{\rho}$	$\log\log \frac{1}{\rho}$	$1/\rho$	$1/\rho$
Time to accuracy ρ :	$n \log \frac{1}{\rho}$	$n \log\log \frac{1}{\rho}$	$1/\rho$	$1/\rho$
Time to excess error ε :	$\frac{1}{\varepsilon^{1/\alpha}} \log^2 \frac{1}{\varepsilon}$	$\frac{1}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon} \log\log \frac{1}{\varepsilon}$	$1/\varepsilon$	$1/\varepsilon$

3.2 Asymptotic analysis of the large-scale case

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. [22]).

A simple application of the uniform convergence results of [25] gives then the upper bound

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

Unfortunately the convergence rate of this bound is too pessimistic. Faster convergence occurs when the loss function has strong convexity properties [9] or when the data distribution satisfies certain assumptions [24]. 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. [13, 4]). Since the three components of the excess error should decrease at the same rate, the solution of the trade-off 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 algorithms 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 the variables n and ρ from the third row results.[2]

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 General recommendations

The rest of this contribution provides a series of recommendations for using stochastic gradient algorithms. Although some of these recommendations seem trivial, experience has shown again and again how easily they can be overlooked.

4.1 Preparing the data

Randomly shuffle the training examples.

Although the theory calls for picking examples randomly, it is usually faster to zip sequentially through the training set. But this does not work if the examples are grouped by class or come in a particular order. Randomly shuffling the examples eliminates this source of problems. Section 1.4.2 provides an additional discussion.

Use preconditioning techniques.

Stochastic gradient descent is a first-order algorithm and therefore suffers dramatically when it reaches an area where the Hessian is ill-conditioned. Fortunately, many simple preprocessing techniques can vastly improve the situation. Sections 1.4.3 and 1.5.3 provide many useful tips.

[2] Note that $\varepsilon^{1/\alpha} \sim \log(n)/n$ implies both $\alpha^{-1}\log\varepsilon \sim \log\log(n) - \log(n) \sim -\log(n)$ and $n \sim \varepsilon^{-1/\alpha}\log n$. Replacing $\log(n)$ in the latter gives $n \sim \varepsilon^{-1/\alpha}\log(1/\varepsilon)$.

4.2 Monitoring and debugging

Monitor both the training cost and the validation error.

Since stochastic gradient descent is useful when the training time is the primary concern, we can spare some training examples to build a decent validation set. It is important to periodically evaluate the validation error during training because we can stop training when we observe that the validation error has not improved in a long time.

It is also important to periodically compute the training cost because stochastic gradient descent is an iterative optimization algorithm. Since the training cost is exactly what the algorithm seeks to optimize, the training cost should be generally decreasing.

A good approach is to repeat the following operations:

1. Zip once through the shuffled training set and perform the stochastic gradient descent updates (4).
2. With an additional loop over the training set, compute the training cost. Training cost here means the criterion that the algorithm seeks to optimize. You can take advantage of the loop to compute other metrics, but the training cost is the one to watch
3. With an additional loop over the validation set, to compute the validation set error. Error here means the performance measure of interest, such as the classification error. You can also take advantage of this loop to cheaply compute other metrics.

Computing the training cost and the validation error represent a significant computational effort because it requires additional passes over the training and validation data. But this beats running blind.

Check the gradients using finite differences.

When the computation of the gradients is slightly incorrect, stochastic gradient descent often works slowly and erratically. This has led many to believe that slow and erratic is the normal operation of the algorithm.

During the last twenty years, I have often been approached for advice in setting the learning rates γ_t of some rebellious stochastic gradient descent program. My advice is to forget about the learning rates and check that the gradients are computed correctly. This reply is biased because people who compute the gradients correctly quickly find that setting small enough learning rates is easy. Those who ask usually have incorrect gradients. Carefully checking each line of the gradient computation code is the wrong way to check the gradients. Use finite differences:

1. Pick an example z.
2. Compute the loss $Q(z, w)$ for the current w.
3. Compute the gradient $g = \nabla_w Q(z, w)$.
4. Apply a slight perturbation $w' = w + \delta$. For instance, change a single weight by a small increment, or use $\delta = -\gamma g$ with γ small enough.
5. Compute the new loss $Q(z, w')$ and verify that $Q(z, w') \approx Q(z, w) + \delta g$.

This process can be automated and should be repeated for many examples z, many perturbations δ, and many initial weights w. Flaws in the gradient computation tend to only appear when peculiar conditions are met. It is not uncommon to discover such bugs in SGD code that has been quietly used for years.

Experiment with the learning rates γ_t using a small sample of the training set.

The mathematics of stochastic gradient descent are amazingly independent of the training set size. In particular, the asymptotic SGD convergence rates [14] are independent from the sample size. Therefore, assuming the gradients are correct, the best way to determine the correct learning rates is to perform experiments using a small but representative sample of the training set. Because the sample is small, it is also possible to run traditional optimization algorithms on this same dataset in order to obtain reference point and set the training cost target.

When the algorithm performs well on the training cost of the small dataset, keep the same learning rates, and let it soldier on the full training set. Expect the validation performance to plateau after a number of epochs roughly comparable to the number of epochs needed to reach this point on the small training set.

5 Linear Models with L_2 Regularization

This section provides specific recommendations for training large linear models with L_2 regularization. The training objective of such models has the form

$$E_n(w) \;=\; \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(y_t w x_t) \tag{10}$$

where $y_t = \pm 1$, and where the function $\ell(m)$ is convex. The corresponding stochastic gradient update is then obtained by approximating the derivative of the sum by the derivative of the loss with respect to a single example

$$w_{t+1} = (1 - \gamma_t \lambda) w_t - \gamma_t y_t x_t \ell'(y_t w_t x_t) \tag{11}$$

Examples:

– Support Vector Machines (SVM) use the non differentiable *hinge loss* [5] :

$$\ell(m) = \max\{0, 1 - m\}\,.$$

– It is often more convenient in the linear case to use the *log-loss*:

$$\ell(m) = \log(1 + e^{-m})\,.$$

The differentiable log-loss is more suitable for the gradient algorithms discussed here. This choice leads to a logistic regression algorithm: probability estimates can be derived using the logistic function:

$$P(y = +1|x) \approx \frac{1}{1 + e^{-wx}}\,.$$

– All statistical models with linear parametrization are in fact amenable to stochastic gradient descent, using the log-likelihood of the model as the loss function $Q(z, w)$. For instance, results for Conditional Random Fields (CRF) [8] are reported in Sec. 5.4.

5.1 Sparsity

Leverage the sparsity of the training examples $\{x_t\}$.
– Represent w_t as a product $s_t W_t$ where $s_t \in \mathbb{R}$.

The training examples often are very high dimensional vectors with only a few non zero coefficients. The stochastic gradient update (11)

$$w_{t+1} = (1 - \gamma_t \lambda) w_t - \gamma_t y_t x_t \ell'(y_t w_t x_t)$$

is then inconvenient because it first rescales *all* coefficients of vector w by factor $(1-\gamma_t\lambda)$. In contrast, the rest of the update only involves the weight coefficients corresponding to a nonzero coefficient in the pattern x_t.

Expressing the vector w_t as the product $s_t W_t$, where s is a scalar, provides a workaround [21]. The stochastic gradient update (11) can then be divided into operations whose complexity scales with the number of nonzero terms in x_t:

$$\begin{aligned} g_t &= \ell'(y_t s_t W_t x_t)\,, \\ s_{t+1} &= (1 - \gamma_t \lambda) s_t\,, \\ W_{t+1} &= W_t - \gamma_t y_t g_t x_t / s_{t+1}\,. \end{aligned}$$

5.2 Learning rates

Use learning rates of the form $\gamma_t = \gamma_0\,(1 + \gamma_0 \lambda t)^{-1}$
– Determine the best γ_0 using a small training data sample.

When the Hessian matrix of the cost function at the optimum is strictly positive, the best convergence speed is achieved using learning rates of the form $(\lambda_{\min} t)^{-1}$ where $\lambda_{\min}$ is the smallest eigenvalue of the Hessian [14]. The theoretical analysis also shows that overestimating $\lambda_{\min}$ by more than a factor two leads to very slow convergence. Although we do not know the exact value of $\lambda_{\min}$, the L_2 regularization term in the training objective function means that $\lambda_{\min} \geq \lambda$. Therefore we can safely use learning rates that asymptotically decrease like $(\lambda t)^{-1}$.

Unfortunately, simply using $\gamma_t = (\lambda t)^{-1}$ leads to very large learning rates in the beginning of the optimization. It is possible to use an additional projection step [21] to contain the damage until the learning rates reach reasonable values. However it is simply better to start with reasonable learning rates. The formula $\gamma_t = \gamma_0(1 + \gamma_0 \lambda t)^{-1}$ ensures that the learning rates γ_t start from a predefined value γ_0 and asymptotically decrease like $(\lambda t)^{-1}$.

The most robust approach is to determine the best γ_0 as explained earlier, using a small sample of the training set. This is justified because the asymptotic SGD convergence rates [14] are independent from the sample size. In order to make the method more robust, I often use a γ_0 slightly smaller than the best value observed on the small training sample.

Such learning rates have been found to be effective in situations that far exceed the scope of this particular analysis. For instance, they work well with nondifferentiable loss functions such as the hinge loss [21]. They also work well when one adds an unregularized bias term to the model. However it is then wise to use smaller learning rates for the bias term itself.

5.3 Averaged Stochastic Gradient Descent

The *averaged stochastic gradient descent* (ASGD) algorithm [19] performs the normal stochastic gradient update (4) and computes the average

$$\bar{w}_t = \frac{1}{t - t_0} \sum_{i=t_0+1}^{t} w_t \,.$$

This average can be computed efficiently using a recursive formula. For instance, in the case of the L_2 regularized training objective (10), the following weight updates implement the ASGD algorithm:

$$\begin{aligned} w_{t+1} &= (1 - \gamma_t \lambda) w_t - \gamma_t y_t x_t \ell'(y_t w_t x_t) \\ \bar{w}_{t+1} &= \bar{w}_t + \mu_t (w_{t+1} - \bar{w}_t) \end{aligned}$$

with the averaging rate

$$\mu_t = 1/\max\{1, t - t_0\} \,.$$

When one uses learning rates γ_t that decrease slower than t^{-1}, the theoretical analysis of ASGD shows that the training error $E_n(\bar{w}_t)$ decreases like t^{-1} with the *optimal constant* [15]. This is as good as the second order stochastic gradient descent (2SGD) for a fraction of the computational cost of (5).

Unfortunately, ASGD typically starts more slowly than the plain SGD and can take a long time to reach the optimal asymptotic convergence speed. Although an adequate choice of the learning rates helps [27], the problem worsens when the dimension d of the inputs x_t increases. Unfortunately, there are no clear guidelines for selecting the time t_0 that determines when we engage the averaging process.

Try averaged stochastic gradient with
– Learning rates $\gamma_t = \gamma_0(1 + \gamma_0\lambda t)^{-3/4}$
– Averaging rates $\mu_t = 1/\max\{1, t-d, t-n\}$

Similar to the trick explained in Sec. 5.1, there is an efficient method to implement averaged stochastic gradient descent for sparse training data. The idea is to represent the variables w_t and $\bar{w}_t$ as

$$\begin{aligned} w_t &= s_t W_t \\ \bar{w}_t &= (A_t + \alpha_t W_t)/\beta_t \end{aligned}$$

where η_t, α_t and β_t are scalars. The average stochastic gradient update equations can then be rewritten in the manner that only involve scalars or sparse operations [27] :

$$\begin{aligned} g_t &= \ell'(y_t s_t W_t x_t)\,, \\ s_{t+1} &= (1 - \gamma_t\lambda)s_t \\ W_{t+1} &= W_t - \gamma_t y_t x_t g_t / s_{t+1} \\ A_{t+1} &= A_t + \gamma_t \alpha_t y_t x_t g_t / s_{t+1} \\ \beta_{t+1} &= \beta_t/(1 - \mu_t) \\ \alpha_{t+1} &= \alpha_t + \mu_t \beta_{t+1} s_{t+1} \end{aligned}$$

Erratum
5/24/2013

The above update rules make sense when $\mu_t < 1$. Since no averaging takes place when $\mu_t = 1$, one should have $\bar{w}_{t+1} = w_{t+1}$. This can be achieved by setting $A_{t+1} = 0$, $\beta_{t+1} = 1$, and $\alpha_{t+1} = s_{t+1}$.

5.4 Experiments

This section briefly reports experimental results illustrating the actual performance of SGD and ASGD on a variety of linear systems. The source code is available at `http://leon.bottou.org/projects/sgd`. All learning rates were determined as explained in section 5.2.

Figure 1 reports results achieved using SGD for a linear SVM trained for the recognition of the CCAT category in the RCV1 dataset [10] using both the hinge loss and the log loss. The training set contains 781,265 documents represented by 47,152 relatively sparse TF/IDF features. SGD runs considerably faster than

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 %

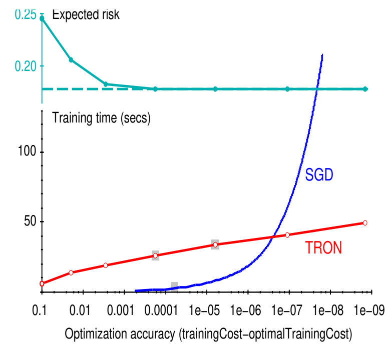

Fig. 1. Results achieved with a L_2 regularized linear model trained on the RCV1 task using both the hinge loss and the log loss. The lower half of the plot shows the time required by SGD and TRON to reach a predefined accuracy ρ on the log loss task. The upper half shows that the expected risk stops improving long before the super-linear optimization algorithm TRON overcomes SGD.

either the standard SVM solvers SVMLight and SVMPerf [7] or the super-linear optimization algorithm TRON [11]

Figure 2 reports results achieved for a linear model trained on the ALPHA task of the 2008 Pascal Large Scale Learning Challenge using the squared hinge loss $\ell(m) = \max\{0, 1-m\}^2$. For reference, we also provide the results achieved by the SGDQN algorithm [1] which was one of the winners of this competition, and works by adapting a separate learning rate for each weight. 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 epoch only.

Figure 3 reports results achieved using SGD, SGDQN, and ASGD for a CRF [8] trained on the CONLL 2000 Chunking task [20]. 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.

6 Conclusion

Stochastic gradient descent and its variants are versatile techniques that have proven invaluable as a learning algorithms for large datasets. The best advice for a successful application of these techniques is (i) to perform small-scale experiments with subsets of the training data, and (ii) to pay a ruthless attention to the correctness of the gradient computation.

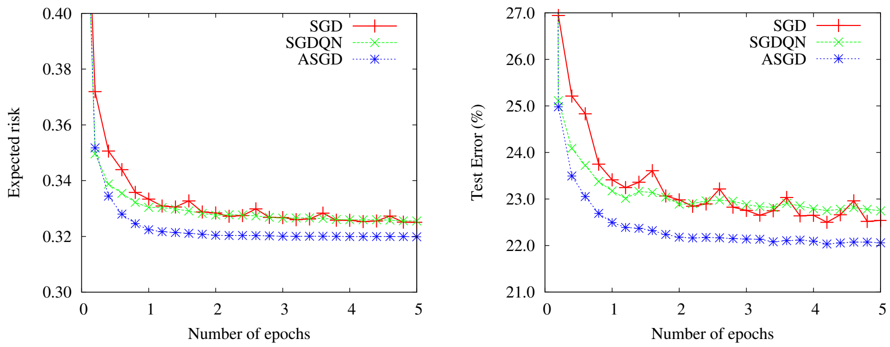

Fig. 2. Comparison of the test set performance of SGD, SGDQN, and ASGD for a L_2 regularized linear model trained with the squared hinge loss on the ALPHA task of the 2008 Pascal Large Scale Learning Challenge. ASGD nearly reaches the optimal expected risk after a single pass.

References

1. Bordes, A., Bottou, L., Gallinari, P.: SGD-QN: Careful quasi-Newton stochastic gradient descent. Journal of Machine Learning Research 10, 1737–1754 (July 2009), with erratum, JMLR 11:2229-2240, 2010
2. Bottou, L., Bousquet, O.: The tradeoffs of large scale learning. In: Platt, J., Koller, D., Singer, Y., Roweis, S. (eds.) Advances in Neural Information Processing Systems, vol. 20, pp. 161–168. NIPS Foundation (http://books.nips.cc) (2008)
3. Bottou, L.: Online algorithms and stochastic approximations. In: Saad, D. (ed.) Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK (1998)
4. Bousquet, O.: Concentration Inequalities and Empirical Processes Theory Applied to the Analysis of Learning Algorithms. Ph.D. thesis, Ecole Polytechnique, Palaiseau, France (2002)
5. Cortes, C., Vapnik, V.: Support-vector network. Machine Learning 20(3), 273–297 (1995)
6. Dennis, J., Schnabel, R.B.: Numerical Methods For Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1983)
7. Joachims, T.: Training linear SVMs in linear time. In: Proceedings of the 12th ACM SIGKDD International Conference. New York (2006)
8. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Brodley, C.E., Danyluk, A.P. (eds.) Proceedings of the Eighteenth International Conference on Machine Learning (ICML). pp. 282–289. Morgan Kaufmann, Williams College, Williamstown (2001)
9. Lee, W.S., Bartlett, P.L., Williamson, R.C.: The importance of convexity in learning with squared loss. IEEE Transactions on Information Theory 44(5), 1974–1980 (1998)
10. Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: RCV1: A new benchmark collection for text categorization research. Journal of Machine Learning Research 5, 361–397 (2004)

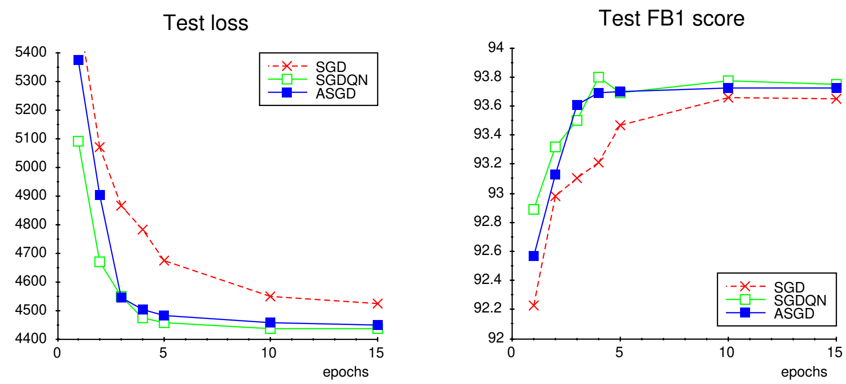

Fig. 3. Comparison of the test set performance of SGD, SGDQN, and ASGD on a L_2 regularized CRF trained on the CONLL Chunking task. On this task, SGDQN appears more attractive because ASGD does not fully reach its asymptotic performance.

11. Lin, C.J., Weng, R.C., Keerthi, S.S.: Trust region newton methods for large-scale logistic regression. In: Ghahramani, Z. (ed.) Proc. Twenty-Fourth International Conference on Machine Learning, (ICML). pp. 561–568. ACM (2007)
12. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: LeCam, L.M., Neyman, J. (eds.) Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probabilities. vol. 1, pp. 281–297. University of California Press, Berkeley and Los Angeles, (Calif) (1967)
13. Massart, P.: Some applications of concentration inequalities to statistics. Annales de la Faculté des Sciences de Toulouse series 6, 9(2), 245–303 (2000)
14. Murata, N.: A statistical study of on-line learning. In: Saad, D. (ed.) Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK (1998)
15. Polyak, B.T., Juditsky, A.B.: Acceleration of stochastic approximation by averaging. SIAM J. Control Optim. 30(4), 838–855 (1992)
16. Robbins, H., Siegmund, D.: A convergence theorem for non negative almost supermartingales and some applications. In: Rustagi, J.S. (ed.) Optimizing Methods in Statistics. Academic Press (1971)
17. Rosenblatt, F.: The perceptron: A perceiving and recognizing automaton. Tech. Rep. 85-460-1, Project PARA, Cornell Aeronautical Lab (1957)
18. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Parallel distributed processing: Explorations in the microstructure of cognition, vol. I, pp. 318–362. Bradford Books, Cambridge, MA (1986)
19. Ruppert, D.: Efficient estimations from a slowly convergent robbins-monro process. Tech. Rep. 781, Cornell University Operations Research and Industrial Engineering (1988)
20. Sang, E.F.T.K., Buchholz, S.: Introduction to the CoNLL-2000 shared task: Chunking. In: Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E.F. (eds.) Proceedings of CoNLL-2000 and LLL-2000. pp. 127–132. Lisbon, Portugal (2000)
21. Shalev-Shwartz, S., Singer, Y., Srebro, N.: Pegasos: Primal estimated subgradient solver for SVM. In: Proc. 24th Intl. Conf. on Machine Learning (ICML'07). pp. 807–814. ACM (2007)

22. Shalev-Shwartz, S., Srebro, N.: SVM optimization: inverse dependence on training set size. In: Proceedings of the 25th International Machine Learning Conference (ICML 2008). pp. 928–935. ACM (2008)
23. Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society (Series B) 58, 267–288 (1996)
24. Tsybakov, A.B.: Optimal aggregation of classifiers in statistical learning. Annals of Statististics 32(1) (2004)
25. Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its Applications 16(2), 264–280 (1971)
26. Widrow, B., Hoff, M.E.: Adaptive switching circuits. In: IRE WESCON Conv. Record, Part 4. pp. 96–104 (1960)
27. Xu, W.: Towards optimal one pass large scale learning with averaged stochastic gradient descent. `http://arxiv.org/abs/1107.2490` (2011)
28. Zinkevich, M.: Online convex programming and generalized infinitesimal gradient ascent. In: Proc. Twentieth International Conference on Machine Learning (2003)