University of California, Berkeley Lecturer: Benjamin Recht

CS227C/STAT260 Convex Optimization and Approximation: Optimization for Modern Data Analysis

Feburary 10, 2015 Notes: Ashia Wilson and Benjamin Recht

Lecture 6: Incremental and Stochastic Gradients

The stochastic gradient method is one of the most popular algorithms for contemporary data analysis and machine learning. It has a long history and has been "invented" several times by many different communities (under the names "least mean squares," "back propagation," "online learning," and the "randomized Kaczmarz method"). Most people attribute this algorithm to the initial work of Robbins and Monro from 1950. There, they were interested in efficient algorithms for computing random means.

In this lecture, we explore some of the properties and implementation details of the stochastic gradient method.

As has been the case, our goal is to minimize $f : \mathbb{R}^d \to \mathbb{R}$. Where the stochastic gradient method differs from the previous methods we have studied is the sort of information we can extract from the function f. Assume that rather than accessing $\nabla f(x)$, we can compute or acquire a random function $g(x)$ such that $\mathbb{E}[g(x)] = \nabla f(x)$. We pretend that g is the gradient and form the update

$$x_{k+1} = x_k - \alpha_k g\left(x_k\right)$$

The intuition behind this method should be clear: we are following a descent direction in expectation, so if we wait long enough, we should be able to get close to the optimal solution. However, it's not quite that simple. Note that if $\nabla f(x_\star) = 0$, then we may move away from $x_\star$ depending on the statistics of $g(x)$. The fact that $x_\star$ is no longer a fixed point complicates the analysis.

1 Noisy gradients

The simplest application of SGM is to the case when we observe noisy gradients. Assume at each iteration, we compute the gradient of a convex function f, but our computation is corrupted by random noise. That is

$$g(x) = \nabla f(x) + \omega$$

where ω is some noise process. The stochastic gradient analysis will allow us to devise a step-size protocol to guarantee convergence to the optimum of $f(x)$ in the presence of such noise.

2 The incremental gradient method

The incremental gradient method, also known as the perceptron or back-propagation, is one of the most common applications of the stochastic gradient method. In this case, we assume that f has the form

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)\,.$$

where n is a very large number. Computing a full gradient requires computation that scales as $O(n)$, but we would like an algorithm that is sublinear in n. The incremental gradient method proceeds by selecting an $i_k \in \{1, \ldots, n\}$ and computing

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) \ .$$

That is, we choose one of the f_i and follow its negative gradient. By cycling through all of the functions, we hope to eventually find a minimizer of f.

This method as presented is not random, but randomness provides extra intuition into why it converges. Note that an unbiased estimate of the gradient of f can be formed by computing

$$g(x) = \nabla f_i(x)$$

where i is selected at random. That is, $\mathbb{E}[g(x)] = \nabla f(x)$. If we select the coordinates in random order, the incremental gradient method becomes a special case of thes stochastic gradient method.

Another reason to use randomness is that the analysis becomes greatly simplified. Though one can show that the incremental gradient method converges, its proof in the randomized case is considerably simpler. Moreover, and perhaps more importantly, the converge guarantees in the non-random case are substantially weaker than the rates in the random case.

2.1 Example: Classification and the Perceptron

As a specific example, let's consider the canonical machine learning problem of classification. We are provided pairs (x_i, y_i), with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ for $i = 1, \ldots n$. The goal is to find a vector $w \in \mathbb{R}^d$ such that

$$w^T x_i > 0 \text{ for } y_i = 1$$
$$w^T x_i < 0 \text{ for } y_i = -1$$

Such a w defines a half-space where we believe all of the positive examples lie on one side and the negative examples on the other.

A popular method was invented in the 50s and uses one example at a time. We initialize our half-space at some w_0. At iteration k, we choose one of our data points, (x_{i_k}, y_{i_k}) and update

$$w_{k+1} = (1 - \gamma) w_k + \eta \begin{cases} y_{i_k} x_{i_k} & y_{i_k} w_k^T x_{i_k} < 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The idea behind this iteration is that if we get the sign incorrect—that is, x_{i_k} lies in the wrong halfspace—then we adjust w_k in a direction to make $w_k^T x_{i_k}$ closer to the correct sign.

It turns out that this is an instance of stochastic gradient descent. A quick calculation will convince you that this procedure is applying the incremental gradient method to the cost function

$$\frac{1}{n} \sum_{i=1}^{n} \max\left(1 - y_i x_i^T w, 0\right) + \lambda \|w\|_2^2 \ . \tag{2}$$

In the update equation (1), we have chosen where $\gamma = \frac{\eta\lambda}{N}$ and η is the stepsize parameter. In machine learning, the stepsize is often referred to as the *learning rate*. The cost function (2) is often called *the support vector machine*. The perceptron algorithm is thus equivalent to "training" a support vector machine using the stochastic gradient method.

2.2 Empirical Risk Minimiation

In machine learning, the Support Vector Machine is one of many instances of the class of optimization problems called *Empirical Risk Minimization*. Many classification, regression, and decision tasks can be evaluated as expected values of error over the data generating distributions. The most common example is the Bayes risk. Given a data generating distribution $p(x, y)$, and a *loss function* $\ell(u, v)$ we define the Bayes risk as

$$R_B[f] := \mathbb{E}[\ell(f(x), y)] \,.$$

This is the expected loss of the decision rule $f(y)$ with respect to the probability distribution $p(x, y)$ that gives rise to the data. The ℓ function measures how much cost we pay for assigning the value $f(x)$ when the quantity to be estimated is y. The goal of many learning tasks is to choose the function f that minimizes the Bayes risk. For example, the Support Vector Machine uses a *hinge loss* that measures how far our prediction $w^T x$ is in to the correct half-space. In regression y is a target variate, and the loss measures the squared difference between $f(x)$ and y.

Often times, minimizing the Bayes risk is computationally intractable and depends strongly on knowing the likelihood and prior models for the data pairs (x, y). A popular alternative uses samples to provided an estimate for the true risk. The setup is as follows, suppose we have a process that generates independent, identically distributed samples $(x_1, y_1), \ldots, (x_N, y_N)$ from the joint distribution $p(x, y)$. Now, for these data points and a fixed decision rule $\hat{x}(y)$, we'd expect that *the empirical risk*

$$R_{\text{emp}}[f] := \frac{1}{N} \sum_{i=1}^{N} \ell(f(y_i), x_i)$$

is "close" to the true Bayes risk. Indeed, $R_{\text{emp}}[f]$ is a random variable equal to the sample mean of the loss function. It is immediate that if we take the expectation with respect to our samples that

$$\mathbb{E}[R_{\text{emp}}[f]] = R_B \,.$$

Now, given these samples, the empirical risk is no longer a function of the likelihood and prior models. Once we condition on the data, we have a simpler optimization problem: minimizing the empirical risk corresponds to finding the best function f that minimizes the average of the loss over our data.

The stochastic gradient method and ERM are intimately tied together. Rather than forming an empirical risk and minimizing this using a method like gradient descent, the SGM would sample a pair (x, y), and then move down the gradient of the loss with respect to the current estimate of f. At the end of the operation, we will have an approximate minimizer of the Bayes Risk. The perceptron algorithm is just a particular instance of this approach to machine learning.

3 Implementation Details

Before we turn to a rigorous analysis of the stochastic gradient method, it will be useful to get some background and insight into how to choose the stepsize parameters. We'll explore the possibilities through some illustrative examples.

3.1 Example: Computing a mean

Consider applying the stochastic gradient method to the function

$$\frac{1}{2n}\sum_{i=1}^{n}(x-\omega_i)^2$$

where ω_i are n fixed scalars. Note that the gradient of one of the increments is

$$\nabla f_i(x) = x - \omega_i \,.$$

Starting with $x_1 = 0$ and use the stepsize $\alpha_k = 1/k$. We can then observe that

$$\begin{aligned}
x_2 &= x_1 - x_1 + \omega_1 = \omega_1 \\
x_3 &= x_2 - \frac{1}{2}(x_2 - \omega_2) = \frac{1}{2}\omega_1 + \frac{1}{2}\omega_2 \\
x_4 &= x_3 - \frac{1}{3}(x_3 - \omega_3) = \frac{1}{3}\omega_1 + \frac{1}{3}\omega_2 + \frac{1}{3}\omega_3
\end{aligned}$$

Thus, we can quickly conclude by induction that

$$x_{k+1} = \left(\frac{k-1}{k}\right)x_k + \frac{1}{k}\omega_k = \frac{1}{k}\sum_{i=1}^{k}\omega_i \,.$$

The $1/k$ stepsize was the originally proposed stepsize by Robbins and Monro. This simple example justifies why: we can think of the stochastic gradient method as computing a running average. Another motivation for the $1/k$ stepsize is that the steps tend to zero, but the path length is infinite.

Note that our cost after n steps is equal to one half of the variance of the sequence $\{\omega_i\}$. In fact, suppose we run the stochastic gradient method on the function

$$f(x) = \tfrac{1}{2}\mathbb{E}[(x-\omega)^2]$$

where ω is some random variable with mean μ and variance σ^2. If we run for n steps with i.i.d. samples of ω at each iteration, the calculation above reveals that

$$x_k = \frac{1}{n}\sum_{i=1}^{n}\omega_i \,.$$

The associated cost is

$$f(x_k) = \tfrac{1}{2}\mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}\omega_i - \omega\right)^2\right] = \frac{1}{2n}\sigma^2 + \tfrac{1}{2}\sigma^2$$

Now, suppose we could just compute the minimizer exactly. In this case, expand $f(x)$ to find

$$f(x) = \tfrac{1}{2}\mathbb{E}[x^2 - 2\omega x + \omega^2] = \tfrac{1}{2}x^2 - 2\mu x + \tfrac{1}{2}\sigma^2 + \tfrac{1}{2}\mu^2 \,.$$

The minimizer is $x_\star = \mu$, and the cost is

$$f(x_\star) = \frac{1}{2}\sigma^2$$

So after n iterations, we have

$$f(x) - f(x_\star) = \frac{1}{2n}\sigma^2 \,.$$

This is the best we could have achieved using any estimator for $x_\star$ given the sequence ω. Interestingly, the "one-at-a-time" or recursive method finds as good a solution as one that considers all of the data together. This example, while trivial, also reveals a fundamental limitation of the stotchastic gradient method: we can't expect to generically get fast convergence rates. *Statistics* not computation, stand in the way of any method achieving linear convergence rates.

Now, one drawback of this example is there was no need for randomness at all. We just marched through the increments in order and converged after n steps to the global optimum. Indeed, if we had chosen a random set of increments, there would be no guarantee that we would have even seen all of the ω_i after n iterations. The next example demonstrates that randomization can dramatically speed up convergence on certain instances.

3.2 Example: The Kaczmarz method

So why do we need randomness? This can be seen by expanding a higher dimensional problem.

$$\min \frac{1}{2n} \sum_{i=1}^{n} \left(a_i^T x - b_i\right)^2$$

Assume that there exists an $x_\star$ such that $a_i^T x_\star = b_i$ for all i, and that $\|a_i\| = 1$. Running the stochastic gradient method with stepsize 1, we find the recursion

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - a_i \left(a_i^T x^{(k)} - b_i\right) \\ &= \left(I - a_i a_i^T\right)\left(x^{(k)} - x_\star\right) + x_\star \\ &= \prod_{i=1}^{k} \left(I - a_i a_i^T\right)\left(x^{(0)} - x_\star\right) + x_\star \end{aligned}$$

We see that we are doing a series of projections onto one dimensional subspaces. If two subsequent subspaces are very close to one another, then we don't make much progress in that particular iteration. Indeed, we can design a sequence of a_i such that we make almost no progress whatsoever.

In contrast, symmetrizing over the order of the product is necessary for noncommutative operators. The following example in fact provides deterministic without-replacement orderings that have exponentially larger norm than the with-replacement expectation. Let $\omega_n = \pi/n$. For $n \geq 3$, define the collection of vectors

$$a_{k;n} = \begin{bmatrix} \cos\left(k\omega_n\right) \\ \sin\left(k\omega_n\right) \end{bmatrix} . \tag{3}$$

Note that all of the $a_{k;n}$ have norm 1 and, for $1 \leq k < n$, $\langle a_{k;n}, a_{k+1;n} \rangle = \cos(\omega_n)$. The matrices $A_k := a_{k;n} a_{k;n}^T$ are all positive semidefinite for $1 \leq k \leq n$, and we have the identity

$$\frac{1}{n} \sum_{k=1}^{n} A_k = \tfrac{1}{2} I \,. \tag{4}$$

Any set of unit vectors satisfying (4) is called a *normalized tight frame*, and the vectors (5) form a *harmonic frame* due to their trigonometric origin.

If we used the stochastic gradient method for n steps, picking steps uniformly at random each time, then

$$\begin{aligned} \mathbb{E}\left[\prod_{i=1}^{n} \left(I - a_{k_i} a_{k_i}^T\right)\left(x^{0)} - x_\star\right)\right] &= \left(I - \frac{1}{n}\sum_{i=1}^{n} a_i a_i^T\right)^n \left(x^{(0)} - x_\star\right) \\ &= 2^{-n}\left(x^{(0)} - x_\star\right) . \end{aligned}$$

Thus, the stocastic gradient method converges *linearly* to the optimal solution. What is different in this example? First, the optimal solution $x_\star$ is a fixed point of both a gradient map *and* a stochastic gradient step. So we don't have to average out the noise that tries to move us away from the optimum.

But the stochastic sampling also contributes to the fast rate of convergence. What happens if we use a deterministic order? Define the vectors

$$\hat{a}_{k;n} = \begin{bmatrix} \sin(-k\omega_n) \\ \cos(-k\omega_n) \end{bmatrix} . \tag{5}$$

Note that

$$I - a_{k;n} a_{k;n}^T = \hat{a}_{k;n} \hat{a}_{k;n}^T$$

Hence, the product of the $I - A_i$ is given by

$$\prod_{i=1}^{k} A_i = \hat{a}_{k;n} \hat{a}_{1;n}^T \prod_{j=1}^{k-1} \langle \hat{a}_{j;n}, \hat{a}_{j+1;n} \rangle = \hat{a}_{k;n} \hat{a}_{1;n}^T \cos^{k-1}(\omega_n) \,,$$

and hence

$$\begin{aligned} \left\| \prod_{i=1}^{k} \left(I - a_i a_i^T\right)\left(x^{(0)} - x_\star\right) \right\| &= \cos^{n-1}(\omega_n) \left| \hat{a}_{1;n}^T \left(x^{(0)} - x_\star\right) \right| \\ &\leq \left(1 - \tfrac{1}{n}\right) \left| \hat{a}_{1;n}^T \left(x^{(0)} - x_\star\right) \right| . \end{aligned}$$

If we had selected $x^{(0)} = [0; 1]$ and $x_\star = 0$, we see that we will have made nearly no progress if we marched through our increments in deterministic order!

Therefore, the arithmetic mean is less than the (deterministic) geometric mean for all $n \geq 3$.

This is a case where picking at random can cause a huge improvement in the convergence rate compared to drawing vectors in some predefined order.

We note here that this particular example has a special name, *the randomized Kaczmarz method*. It is a simple case of the stochastic gradient method for solving least-squares. This method was analyzed by signal processing researchers, without knowledge of the long standing work on the stochastic gradient method.

3.3 Epochs

Another central concept in stochastic gradient methods is the notion of *epochs*. In an epoch, some number of iterations are run, and then a choice is made about whether to change the stepsize. A common strategy is to run with a constant step size for some fixed number of iterations T, and then reduce the stepsize by a constant factor γ. Thus, if our initial stepsize is α, on the kth epoch, the stepsize is $\alpha\gamma^{k-1}$. This method is often more robust in practice than the diminishing stepsize rule. For this stepsize rule, a reasonable heuristic is to choose γ between 0.8 and 0.9.

Another rule which we will study is called *epoch doubling*. In epoch doubling, we run for T steps with stepsize α, then run $2T$ steps with stepsize $\alpha/2$, and then $4T$ steps with stepsize $\alpha/4$ and so on. Note that this provides a piecewise constant approximation to the function α/k.

3.4 Momentum

Finally, we note that one can run stochastic gradient descent with momentum. This method would be identical to the momentum methods studied earlier in the course, with the gradient replaced by a stochastic gradient. That is:

$$x_{k+1} = x_k - \alpha_k g\left(x_k\right) + \beta(x_k + x_{k-1})\,.$$

In practice, these methods are very successful. Typical choices for β here are between 0.8 and 0.95. The theoretical guarantees for momentum methods only demonstrate meager gains over the standard SGM. Essentially, we know that the function value will converge at a rate of $1/k$, but the constant in front of the $1/k$ can be made smaller using momentum or acceleration. Regardless of the theoretical guarantees, one should always keep in mind that momentum can provide significant accelerations, and should be considered an option in any implementation of SGM.