COS 324, Precept #7: More on Stochastic Gradient Descent

November 10, 2017

1 Overview

In this class, we've covered two closely related algorithms for convex optimization: gradient descent and stochastic gradient descent. When machine learning appears in the news these days (AlphaGo, self-driving cars, end-to-end machine translation, inverse image search, ...), SGD is the underlying optimization algorithm in virtually *every* case. As such, this precept will be about gaining a deeper intuition for this important algorithm.

2 Review of GD and SGD

Recall that the gradient descent update ("roll the ball down the $f(x)$-shaped hill") is given by:

$$x_{t+1} \leftarrow x_t - \eta \nabla f(x_t).$$

If f is convex and Lipschitz, recall that gradient descent (with $\eta = \frac{D}{G\sqrt{T}}$) converges:

$$\frac{\sum_{t=1}^{T} f(x_t)}{T} \leq \min_x f(x) + \frac{DG}{\sqrt{T}}.$$

Recall that SGD runs gradient descent steps with a *stochastic gradient*, an unbiased estimator $\widehat{\nabla} f$ for the gradient. that is,

$$\mathbb{E}[\widehat{\nabla} f(x)] = \nabla f(x).$$

Then, we have convergence in expectation (over the randomness of the iterates x_t):

$$\mathbb{E}\left[f(\overline{x}_T)\right] \leq \mathbb{E}\left[\frac{\sum_{t=1}^{T} f(x_t)}{T}\right] \leq \min_x f(x) + \frac{DG}{\sqrt{T}},$$

where G is now an upper bound on the magnitude of $\widehat{\nabla} f$.

An important case is when the loss function $f(x)$ is an average of loss functions, as in many empirical risk minimization problems:

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

a widely-considered gradient estimator is $\nabla f_j(x)$, for a j sampled uniformly at random. This case, sometimes known as the *finite-sum* case, is one of the *raisons d'être* of SGD.

3 Example: Linear regression with the Huber loss

Let's go through an end-to-end example of accelerating regression using SGD. We'll consider the problem of linear regression with the Huber loss, the outlier-tolerant convex loss function defined by pasting together a quadratic function near 0 with linear functions at the ends:

$$\ell(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \le 1 \\ |x| - \frac{1}{2} & \text{otherwise} \end{cases}$$

We can take its derivative:

$$\ell'(x) = \begin{cases} x & \text{if } x \le 1 \\ \operatorname{sgn}(x) & \text{otherwise} \end{cases}$$

Now, recall that in linear regression, we have m data points $(x_i \in \mathbb{R}^d, y_i \in \mathbb{R})$, and we'd like to find a linear function $w \in \mathbb{R}^d$ such that $w^\top x_i \approx y_i$. We'd like to minimize the average Huber loss

$$F(w) = \frac{1}{m} \sum_{i=1}^{m} \ell(w^\top x_i - y_i).$$

Using the chain rule, we can compute the gradient of F:

$$\begin{aligned} \nabla F(w) &= \frac{1}{m} \sum_{i=1}^{m} \nabla \ell(w^\top x_i - y_i) = \frac{1}{m} \sum_{i=1}^{m} x_i \cdot \ell'(w^\top x_i - y_i) \\ &= \frac{1}{m} \sum_{i=1}^{m} \begin{cases} (w^\top x_i - y_i)\, x_i & \text{if } |w^\top x_i - y_i| \le 1 \\ \operatorname{sgn}(w^\top x_i - y_i)\, x_i & \text{otherwise} \end{cases} \end{aligned}$$

This is a formula for the full gradient of the loss, with respect to the model parameters w. Let's analyze the time complexity of running gradient descent with this gradient.

Each iteration, we must do $O(md)$ work: an n-dimensional inner product for each of m data points. The convergence theorem of gradient descent states that G^2D^2/ε^2 iterations suffice to reach an ε-approximate global minimum. We can treat G and D as constants, as long as the data are normalized ($\|x_i\| \le C_1$) and the solution isn't too large ($\|w^*\| \le C_2$).

Thus, the total time complexity is $O(md/\varepsilon^2)$.

3.1 Replacing the gradient with its estimator

Now, we consider SGD for the same problem. There is an efficient routine for estimating $\nabla F(w)$: pick a data point j uniformly at random, and return only the j-th summand of the full gradient. Indeed, by definition of expectation:

$$\mathbb{E}[\widehat{\nabla} F(w)] = \sum_{i=1}^{m} \underbrace{\Pr[j = i]}_{1/m} \cdot \nabla \ell(w^\top x_i - y_i) = \nabla F(w).$$

Using this estimator, each step of SGD takes $O(d)$ time instead of $O(md)$; it only looks at a single data point! Noting that G and D are the same as before, we get the same convergence guarantee as gradient descent *in expectation*, with much (factor of m) cheaper iterations.

4 Variance: the problem with SGD

This acceleration afforded by SGD doesn't come for free. The stochastic gradient steps introduce uncertainty to the procedure; moreover, this uncertainty is accumulated over T steps. One way to quantify this is to consider the *variance* of the estimator:[1]

$$\mathrm{Var}[\widehat{\nabla} f(x)] = \mathbb{E}[\|\widehat{\nabla} f(x) - \nabla f(x)\|^2].$$

Consider the sources of uncertainty on the final averaged output of SGD:

$$\begin{aligned} \overline{x}_T &:= \frac{1}{T}\left(x_1 + x_2 + \ldots + x_T\right) \\ &= x_1 - \frac{T-1}{T}\eta \widehat{\nabla} f(x_1) - \frac{T-2}{T}\eta \widehat{\nabla} f(x_2) - \ldots - \frac{\eta}{T}\widehat{\nabla} f(x_{T-1}). \end{aligned}$$

In the worst case, $\mathrm{Var}[\widehat{\nabla} f(x)]$ can be as large as G^2. Recalling the choice of $\eta = \frac{D}{G\sqrt{T}}$, we have

$$\mathrm{Var}[\overline{x}_T] \le \frac{1^2 + 2^2 + \ldots + (T-1)^2}{T^2} \cdot \eta^2 G^2 \approx D^2/3.$$

This is bad news: the iterates of SGD can vary on the scale of D, the total distance travelled by the algorithm. Although this worst case seldom occurs in practice, it is often the case that SGD faces a bottleneck on account of its uncertainty. In both theory and practice, we would like to *stabilize* SGD.

[1] Beware: this is a scalar quantity associated with a vector-valued random variable.

4.1 Strategies to reduce variance

(This part can be an open-ended discussion.)

One very practical way to reduce the variance of the stochastic gradients is to employ *mini-batching*: instead of sampling a single summand in the loss function (the loss induced by a single data point), sample a random subset of points S. Then, use the gradient estimator

$$\widehat{\nabla} f(x) = \frac{1}{|S|} \sum_{i \in S} \nabla f_i(x).$$

This reduces the variance of the gradients by a factor of $|S|$, at the cost of a factor of $|S|$ running time per iteration, compared to vanilla SGD. Going back to the regression scenario, we consider at each iteration a randomly sampled *mini-batch* of data points, and take the gradient of the loss with respect to this mini-batch (rather than a single point).

You can think of mini-batching as a way to interpolate between SGD ($|S| = 1$) and full gradient descent ($|S| = T$). As $|S|$ is increased, the batches become more representative of the entire dataset, so that the stochastic gradients become increasingly precise estimators of the full gradient. Often, $|S|$ is chosen in a hardware-aware way (e.g. the right number of data points to fill a high-performance cache).

There are other ways to reduce the variance of SGD– for example, the aptly-named *stochastic variance-reduced gradient* (SVRG) algorithm queries single gradients (∇f_j) like SGD, but computes a full gradient every so often, enabling the construction of better estimators for the iterations in between full-gradient snapshots. To analyze such algorithms is outside the scope of the class, but it is worth mentioning that variance reduction continues to be an active research topic.

5 When variance is intentional

(This part can also be an open-ended discussion.)

A final note: sometimes, the variance of SGD is a *desired* property. Sometimes, rather than finding the minimum of a convex function f, we'd like to *sample* points near the minimum of f. Then, we can run (full) gradient descent, but *corrupt* the iterates with Gaussian noise vectors. This is better viewed as a Monte Carlo sampling algorithm (rather than an optimization method), which goes by the name of *Langevin dynamics.*

Another scenario is when the loss function is non-convex, in which case adding uncertainty to an optimization algorithm may help in escaping local minima or saddle points. Indeed, when training neural networks with SGD (a non-convex optimization problem), selecting mini-batch sizes and learning rates is a delicate matter of balancing this phenomenon with the conflicting objective of stability.