10-425/625: Introduction to Convex Optimization **(Fall 2023)**

Lecture 12: Stochastic Gradient Descent

Instructor:[1] *Matt Gormley* *October 4, 2023*

Today we'll talk about the stochastic gradient descent (SGD) algorithm.

12.1 Stochastic Gradient Descent

SGD has a long, rich history and the basic algorithm has been reinvented many times. The algorithm in roughly the form we study it today is usually attributed to Robbins and Munro who were trying to find roots of functions with noisy function access (very similar to function optimization, with noisy gradient access).

As usual we're trying to minimize a function f (for now, lets suppose that f is differentiable, and that we're interested in solving an unconstrained problem). In many examples (we'll see several in this lecture), it will be natural to hypothesize that rather than obtaining the exact gradient value $\nabla f(x)$, at some point x, we are able to compute a vector $g(x, \xi)$ which is a function of a random variable ξ and has the property that,

$$\mathbb{E}_\xi[g(x, \xi)] = \nabla f(x).$$

We'll generally suppose that ξ has distribution P. The expectation above is saying that the stochastic gradient $g(x, \xi)$ is an unbiased estimate of the actual gradient $\nabla f(x)$.

The SGD algorithm then simply follows the iterates:

$$x^{t+1} = x^t - \eta_t g(x^t, \xi^t),$$

where ξ^t has distribution P, and is drawn independently of everything else. Often computing $g(x^t, \xi^t)$ will be much faster than computing $\nabla f(x^t)$, but in general it can be a very noisy estimate of $\nabla f(x^t)$ (i.e. it might for instance have high variance).

[1]These notes were originally written by Siva Balakrishnan for 10-725 Spring 2023 (original version: here) and were edited and adapted for 10-425/625.

Similar to the subgradient method we'll need to be careful about our choice of step-size, i.e. for instance it's easy to see that (unlike for GD in the smooth case) fixed step-size choices don't usually work. If we're at the optimum x^*, the gradient might be zero, but the stochastic gradient might still be non-zero (variance), and we'll need to carefully choose the step-size to decay to ensure convergence.

12.2 Some Examples of SGD Algorithms

12.2.1 Noisy Gradients

Maybe the most intuitive setting is where rather than have access to gradients, we have access to a noisy oracle, i.e. we can make measurements of the gradient which are corrupted by additive noise, i.e.

$$g(x, \xi) = \nabla f(x) + \xi,$$

where $\mathbb{E}[\xi] = 0$.

12.2.2 Incremental Gradient Method

This is the idea behind many algorithms (including things like backpropogation), where we are attempting to minimize a function:

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

Computing the gradient of f then requires, computing the gradient of each of $f_1, \ldots, f_n$, and the incremental gradient method instead just cycles through the functions, using the updates:

$$x^{t+1} = x^t - \eta_t \nabla f_{i^t}(x^t),$$

where $i^t = ((t-1) \bmod n) + 1$. This isn't really an SGD algorithm (and can be reasonably difficult to analyze).

12.2.3 Randomized Incremental Gradient Method

Instead imagine the randomized version – at each iteration we choose an index i^t uniformly at random from $\{1, \ldots, n\}$, and we repeat the same iteration as above.

Here ξ^t denotes the (random) choice of index, and $g(x, \xi^t) = \nabla f_{\xi^t}(x)$. It's easy to check that,

$$\mathbb{E}[g(x, \xi)] = \nabla f(x),$$

so this randomized variant is indeed an SGD algorithm.

12.2.4 Incremental Gradient Method (IGM) with Random Permutations

In practice, the method above might run into a problem: we might never see one of the functions f_j after some number of iterations. That is, the random variable i^t might never equals j — this is a low probability event, but it would equate to throwing out training examples in machine learning. So we'd really like to avoid it.

The typical SGD implementation follows the IGM, but at each epoch (i.e. pass through the the intergers $1, \ldots, n$) first takes some random permutation of them and follows that order instead. That is, if the function shuffle(l) returns a random permutation of l. Then this version does as follows:

```
$s = 0$
for $t \in 1, \ldots, T$:
    for $i \in$ shuffle$(1, \ldots, n)$:
        $x^{s+1} = x^s - \eta_s \nabla f_i(x^s)$
        $s = s + 1$
```

This is just like Randomized IGM except that where it does sampling *with* replacement, this version does sampling *without* replacement of the integers $1, \ldots, n$. The result is that in each epoch we see each f_i exactly once.

12.2.5 SGD with Momentum

Here we briefly note that another useful trick for dealing with nonconvexity is to include a momentum term in SGD. The idea is that we include an additional weight $\beta \in [0, 1]$ that trades off between how much to step in the direction of the current gradient (small β) and how much to continue moving the direction that we have been moving (large β). In short, $\delta^{(t)}$ keeps an exponential moving average of the gradient vectors.

$$\delta^{(t)} = \beta\delta^{(t-1)} + (1-\beta)\nabla f_{i^t}(x^t)$$
$$\theta^{(t)} = \theta^{(t-1)} - \eta\delta^{(t)}$$

In practice, we usually implement this within the IGM with Random Permutations setup, shuffling the order in which we visit each f_i at the start of each epoch.

(We will discuss this algorithm in more detail later in the course and explore it further in the homework.)

12.3 ERM and Population Risk Minimization

Empirical Risk Minimization is one of the standard lenses through which we come up with algorithms (and statistical analysis) in machine learning. The so-called "statistical learning" setup, is that we have a loss function, and an associated risk, i.e. the expected loss. Our goal, broadly speaking, is to find rules/functions/... which have small risk (i.e. small expected loss) given some "training samples" from a distribution.

We are typically given samples $\{(X_1, y_1), \ldots, (X_n, y_n)\} \sim P_{Xy}$ and given a rule/classifier/regressor f, we evaluate it via:

$$R(f) = \mathbb{E}_{X,y\sim P_{Xy}}[\ell(f(X), y)],$$

where ℓ measures the loss for making a prediction $f(X)$ when the true label is y.

We cannot directly minimize R to find the best rule since we don't have access to the distribution P_{Xy} so a standard idea is to instead attempt to minimize:

$$R_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(X_i), y_i),$$

over candidate functions f.

One can now see at least two connections to SGD:

1. We can apply the randomized incremental gradient algorithm to the *empirical risk*. Usually our rules f will be parametrized by some parameters, and we'll be doing SGD on those parameters but for now the exact details are not so important.

2. Perhaps less obviously – if we only make one pass over the training samples, then we can view the incremental gradient algorithm as *directly* minimizing the *population risk*, i.e. $\mathbb{E}[\nabla \ell(f(X_i), y_i)] = \nabla R(f)$.

   This connection is very interesting, i.e. if we can show that one-pass SGD works in this setting (i.e. converges to something close to the minimizer of $R(f)$ at some rate) – then we can obtain "generalization bounds" directly.

To summarize, in the "statistical learning" setting one can view SGD either as an algorithm for minimizing the empirical risk (in which case, we'll need to separately reason about how good the ERM solution is, i.e. separately prove a generalization bound). Alternatively, one can view it as an algorithm for minimizing the population risk directly.