Lecture 9: Diminishing step sizes and accelerating SGD with averaging.

CS4787 — Principles of Large-Scale Machine Learning Systems

Recap: Over the last week, we looked at a couple of ways to accelerate learning by using techniques that help decrease the dependence on the condition number of the problem κ. One of these methods was AdaGrad, which we saw had the side effect of decreasing the step size over time. Doing this will cause the algorithm to converge asymptotically towards having low gradient magnitude, unlike plain constant-step-size SGD which converges to a noise ball.

Recall that from the Lecture 4 notes, we had that (for strongly convex SGD)

$$\mathbf{E}\left[f(w_T) - f^*\right] \leq \exp(-\mu\alpha T) \cdot (f(w_0) - f^*) + \frac{\alpha\sigma^2\kappa}{2}.$$

Last week, we looked at the effect of the κ term on this and other algorithms; today, we'll look at the effect of the α and σ^2 parts of the noise ball, and discuss some ways to address the effect of these to accelerate SGD.

Diminishing step size schemes. Since α is a parameter we can just set, a natural way of trying to "get at" this term is to just decrease the value of α over time. This involves a *time-varying* step size, resulting in an algorithm with update step

$$w_{t+1} = w_t - \alpha_t \nabla f_{\tilde{i}_t}(w_t).$$

(The form for minibatched SGD is analogous.) **Problem: naively, each each α_t is a separate hyperparameter we'd need to tune...but choosing so many hyperparameters is prohibitive. Can we get some better rintuition for how fast we should decrease the learning rate?**

An example: strongly convex SGD. From the Lecture 4 notes, we had that for a single-step update of (μ-strongly convex, L-Lipschitz gradient) SGD with step size $\alpha \leq 1/L$,

$$\mathbf{E}\left[f(w_{t+1}) - f^*\right] \leq (1 - \mu\alpha)\mathbf{E}\left[f(w_t) - f^*\right] + \frac{\alpha^2\sigma^2 L}{2}.$$

It follows by the same argument that if instead we use the time varying step size α_t,

$$\mathbf{E}\left[f(w_{t+1}) - f^*\right] \leq (1 - \mu\alpha_t)\mathbf{E}\left[f(w_t) - f^*\right] + \frac{\alpha_t^2\sigma^2 L}{2}.$$

How should we choose α_t? Well, to gain intuition, imagine that we somehow *know* what $\mathbf{E}\left[f(w_t) - f^*\right]$ is. What value of α_t should we choose to minimize our bound on the objective gap at the next timestep? To find this, we can differentiate to minimize, which gives us

$$0 = -\mu\mathbf{E}\left[f(w_t) - f^*\right] + \alpha_t\sigma^2 L \quad \Rightarrow \quad \alpha_t = \frac{\mu\mathbf{E}\left[f(w_t) - f^*\right]}{\sigma^2 L}.$$

If we set α_t in this way (assuming for the moment that it doesn't result in α_t being greater than $1/L$), then

$$\begin{aligned}\mathbf{E}\left[f(w_{t+1}) - f^*\right] &\leq \left(1 - \mu \cdot \frac{\mu \mathbf{E}\left[f(w_t) - f^*\right]}{\sigma^2 L}\right) \mathbf{E}\left[f(w_t) - f^*\right] + \frac{\sigma^2 L}{2}\left(\frac{\mu \mathbf{E}\left[f(w_t) - f^*\right]}{\sigma^2 L}\right)^2 \\ &\leq \mathbf{E}\left[f(w_t) - f^*\right] - \frac{\mu^2 \mathbf{E}\left[f(w_t) - f^*\right]^2}{2\sigma^2 L}\end{aligned}$$

Now, we're going to use a little algebra trick: since

$$x^2 \geq x^2 - y^2 = (x+y)(x-y) \quad \Rightarrow \quad \frac{1}{x+y} \geq \frac{x-y}{x^2} = \frac{1}{x} - \frac{y}{x^2},$$

it follows that if we invert this whole expression, then

$$\begin{aligned}\mathbf{E}\left[f(w_{t+1}) - f^*\right]^{-1} &\geq \left(\mathbf{E}\left[f(w_t) - f^*\right] - \frac{\mu^2 \mathbf{E}\left[f(w_t) - f^*\right]^2}{2\sigma^2 L}\right)^{-1} \\ &\geq \mathbf{E}\left[f(w_t) - f^*\right]^{-1} + \frac{\mu^2 \mathbf{E}\left[f(w_t) - f^*\right]^2}{2\sigma^2 L \cdot \mathbf{E}\left[f(w_t) - f^*\right]^2} \\ &= \mathbf{E}\left[f(w_t) - f^*\right]^{-1} + \frac{\mu^2}{2\sigma^2 L}.\end{aligned}$$

Now by induction it follows immediately that

$$\mathbf{E}\left[f(w_T) - f^*\right]^{-1} \geq (f(w_0) - f^*)^{-1} + \frac{\mu^2 T}{2\sigma^2 L}.$$

Inverting again gives us

$$\mathbf{E}\left[f(w_T) - f^*\right] \geq \frac{2\sigma^2 L (f(w_0) - f^*)}{2\sigma^2 L + \mu^2 \cdot (f(w_0) - f^*) \cdot T}$$

which corresponds to a setting of α_T of

$$\alpha_T = \frac{\mu \mathbf{E}\left[f(w_T) - f^*\right]}{\sigma^2 L} = \frac{2\mu (f(w_0) - f^*)}{2\sigma^2 L + \mu^2 \cdot (f(w_0) - f^*) \cdot T}.$$

Observe that this varying learning rate/step size is of the form

$$\alpha_T = \frac{\alpha_0}{1 + \eta T},$$

which we sometimes call a $1/T$ step size scheme. This is a nice way to parameterize a diminishing step size scheme, requiring only two hyperparameters, α_0 and η (rather than one for each iteration). Of course, for this to satisfy our assumption we need $\alpha_0 \leq 1/L$. We could go back and derive a rate here, but I won't do this...the point is to give you intuition for *why* a $1/T$ rate makes sense for strongly convex optimization.

Other options:

- For non-convex optimization and non-strongly-convex optimization, often a $1/T$ step size rate is decreasing too quickly. Here, we often adopt a $1/\sqrt{T}$ step size rate that is structured similarly.
- We can also decrease the step sizes infrequently in *epochs*, rather than at every iteration.
- For some tasks (e.g. some deep learning architectures), it makes sense to *increase* the step size first, and then decrease it. For example, if we expect to move from a region of very high noise early in the optimization to a region of much lower noise later, this could make sense. This approach, especially when repeated, is related to *annealing* and is sometimes called that.

Increasing minibatch sizes. Another thing we can try to do to target the variance term is to use a minibatch size that becomes larger over time. Here, we'll look at how this can effect convergence in the non-convex optimization setting (just to vary our setting from our diminishing step size case, not because either method is particularly suited to either setting).

If we use a constant learning rate and a minibatch of size B_t at time t, then (from the analysis we did in Lecture 4), we would get

$$\mathbf{E}\left[f(w_{t+1})\right] \leq \mathbf{E}\left[f(w_t)\right] - \frac{\alpha}{2} \cdot \mathbf{E}\left[\|\nabla f(w_t)\|^2\right] + \underbrace{\frac{\alpha^2 L}{2} \cdot \mathbf{E}\left[\left\|\frac{1}{B_t}\sum_{b=1}^{B_t} \nabla f_{\tilde{i}_{b,t}}(w_t) - \nabla f(w_t)\right\|^2\right]}_{\text{second-order variance/error term}}.$$

If (as we usually assume) the variance of an individual example gradient is bounded by σ^2 as

$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(w_t) - \nabla f(w_t)\|^2 \leq \sigma^2,$$

then by the analysis we've already done for minibatch SGD we know that

$$\mathbf{E}\left[f(w_{t+1})\right] \leq \mathbf{E}\left[f(w_t)\right] - \frac{\alpha}{2} \cdot \mathbf{E}\left[\|\nabla f(w_t)\|^2\right] + \frac{\alpha^2 \sigma^2 L}{2B_t}$$

which implies that

$$\frac{\alpha}{2} \cdot \mathbf{E}\left[\|\nabla f(w_t)\|^2\right] \leq \mathbf{E}\left[f(w_t)\right] - \mathbf{E}\left[f(w_{t+1})\right] + \frac{\alpha^2 \sigma^2 L}{2B_t}.$$

Summing this up over T iterations of SGD, and telescoping the sum as usual, we get

$$\begin{aligned}\frac{\alpha}{2}\sum_{t=0}^{T-1} \mathbf{E}\left[\|\nabla f(w_t)\|^2\right] &\leq \mathbf{E}\left[f(w_0)\right] - \mathbf{E}\left[f(w_T)\right] + \sum_{t=0}^{T-1} \frac{\alpha^2 \sigma^2 L}{2B_t} \\ &\leq f(w_0) - f^* + \sum_{t=0}^{T-1} \frac{\alpha^2 \sigma^2 L}{2B_t}\end{aligned}$$

which implies that

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbf{E}\left[\|\nabla f(w_t)\|^2\right] \leq \frac{2(f(w_0) - f^*)}{\alpha T} + \frac{\alpha \sigma^2 L}{T}\sum_{t=0}^{T-1} \frac{1}{B_t}.$$

This means that any increasing batch size scheme is going to converge, and if the sum of the reciprocals of the batch sizes converges, then SGD with this scheme will converge at a rate of $1/T$.

We can do a similar analysis for convex problems...I won't discuss this in class but there's a great analysis in Chapter 5 of "Optimization Methods for Large-Scale Machine Learning" if you are curious.

Polyak averaging. Intuition: SGD is converging to a "noise ball" where the iterates are randomly jumping around some space surrounding the optimum. We can think about these iterates as random samples that approximate the optimum.

What can we do when we have a bunch of random samples that approximate something to improve the precision of our estimate?

Technique: run regular SGD and just average the iterates. That is,

$$w_{t+1} = w_t - \alpha_t \nabla f_{\tilde{i}_t}(w_t)$$
$$\bar{w}_{t+1} = \frac{t}{t+1} \cdot \bar{w}_t + \frac{1}{t+1} \cdot w_{t+1};$$

eventually we output the average $\bar{w}_T$ at the end of execution. This is equivalent to writing

$$\bar{w}_T = \frac{1}{T} \sum_{t=1}^{T} w_t.$$

The main idea here is that we're averaging out a bunch of iterations w_t that are all in the noise ball. When you average out a bunch of noisy things, often you are able to reduce the noise.

One issue with this is that we are averaging with equal weight iterates from the very start of training, when we have not reached the noise ball. In order to address this, we often run averaged SGD by first using **a warm-up period** during which we do not average, and then only starting to average after the warm-up period is over. Long warm-up periods (such as half of the total number of epochs ran) usually produce better results in practice than averaged SGD without any warmup.

For **Polyak averaging on non-convex problems**, we run into the same issue that we ran into with AdaGrad: as we move through a non-convex landscape, we may get very far away from the parameter values we were at during previous iterations. If this happens, averaging together with those parameter values doesn't really make sense, and can hurt the performance of our algorithm. So, instead, a standard approach is to use an *exponential moving average*, just like was done to transform AdaGrad into RMSProp. That is, we pick some decay factor $0 < \rho < 1$ and run

$$w_{t+1} = w_t - \alpha_t \nabla f_{\tilde{i}_t}(w_t)$$
$$\bar{w}_{t+1} = \rho \cdot \bar{w}_t + (1-\rho) \cdot w_{t+1}.$$

This running average approach often outperforms other averaging methods in non-convex settings.

Stochastic variance-reduced gradient. Idea: reduce the variance of the gradient estimators by using an infrequent full-gradient step.

`Procedure SVRG`

Parameters update frequency m and learning rate η
Initialize $\tilde{w}_0$
Iterate: for $s = 1, 2, \ldots$
  $\tilde{w} = \tilde{w}_{s-1}$
  $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \nabla \psi_i(\tilde{w})$
  $w_0 = \tilde{w}$
  Iterate: for $t = 1, 2, \ldots, m$
    Randomly pick $i_t \in \{1, \ldots, n\}$ and update weight
      $w_t = w_{t-1} - \eta(\nabla \psi_{i_t}(w_{t-1}) - \nabla \psi_{i_t}(\tilde{w}) + \tilde{\mu})$
  end
  option I: set $\tilde{w}_s = w_m$
  option II: set $\tilde{w}_s = w_t$ for randomly chosen $t \in \{0, \ldots, m-1\}$
end