Lecture 7: Accelerating SGD with Momentum

CS4787 — Principles of Large-Scale Machine Learning Systems

Recall: When we analyzed gradient descent and SGD for strongly convex objectives, the convergence rate depended on the *condition number* $\kappa = L/\mu$. For example, the noise ball size for SGD with a constant step size was determined by

$$\mathbf{E}\left[f(w_T) - f^*\right] \leq (1-\alpha\mu)^T \cdot (f(w_0) - f^*) + \frac{\alpha\sigma^2 L}{2\mu} = (1-\alpha\mu)^T \cdot (f(w_0) - f^*) + \frac{\alpha\sigma^2\kappa}{2}$$

and the convergence rate for gradient descent with constant step size was

$$f(w_T) - f^* \leq \exp\left(-\frac{\mu T}{L}\right) \cdot (f(w_0) - f^*) = \exp\left(-\frac{T}{\kappa}\right) \cdot (f(w_0) - f^*)\,.$$

Takeaway: when the condition number is high, convergence can become slow. Motivates the question: How can we speed up gradient descent when the condition is high?

To understand this more easily, let's consider the simplest possible setting with a high condition number: a two-dimensional quadratic. Specifically,

$$f(w) = f(w_1, w_2) = \frac{L}{2}w_1^2 + \frac{\mu}{2}w_2^2 = \frac{1}{2}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^T \begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

Here, the optimum value occurs at $w^* = 0$, and the second derivative matrix is the constant

$$\nabla^2 f(w) = \begin{bmatrix} L & 0 \\ 0 & \mu \end{bmatrix},$$

and the eigenvalues of this matrix are L and μ, so if $L \geq \mu > 0$ then f is strongly convex with strong convexity constant μ and its second derivative is bounded by L. If we think about what gradient descent (with constant learning rate) is going to do on this problem, its update rule will be the following

$$\begin{aligned}(w_{t+1})_1 &= (w_t)_1 - \alpha L (w_t)_1 \\ (w_{t+1})_2 &= (w_t)_2 - \alpha\mu(w_t)_2.\end{aligned}$$

This means that

$$\begin{aligned}\left((w_T)_1\right)^2 &= (1-\alpha L)^{2T}\left((w_0)_1\right)^2 \\ \left((w_T)_2\right)^2 &= (1-\alpha\mu)^{2T}\left((w_0)_2\right)^2\end{aligned}$$

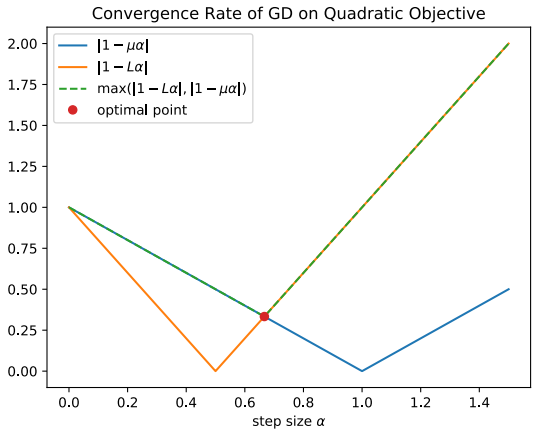
and so

$$f(w_T) = \frac{L}{2}(1-\alpha L)^{2T}\left((w_0)_1\right)^2 + \frac{\mu}{2}(1-\alpha\mu)^{2T}\left((w_0)_2\right)^2.$$

Asymptotically, this will be dominated by whichever term grows slower. That is

$$f(w_T) = \mathcal{O}\left(\max(|1-\alpha L|, |1-\alpha\mu|)^{2T}\right).$$

If we look at this inner maximum as a function of α, a curious effect emerges.



So while we would *want* to pick a smaller learning rate for the L term, and a larger learning rate for the μ term, we're forced to pick something in the middle. (Incidentally, this optimal learning rate occurs at

$$\alpha = \frac{2}{L+\mu} \quad \Rightarrow \quad \max(|1-\alpha L|, |1-\alpha\mu|) = \frac{L-\mu}{L+\mu} = \frac{\kappa-1}{\kappa+1} = 1 - \frac{2}{\kappa+1}$$

but it's not super important to know this.) That is, wed like to *set the step size larger for dimension with less curvature, and smaller for the dimension with more curvature*. But we can't do this with plain GD or SGD, because there is only one step-size parameter.

What can we do about this problem? Three common solutions:

- Momentum (this lecture)
- Adaptive learning rates (next week)
- Preconditioning (next week)

Momentum. Motivation: try to tell the difference between more and less curved directions using information already available in gradient descent. Idea: in the one-dimensional case, if the gradients **are reversing sign**, then the step size is too large, because we're overshooting the optimum. Conversely, if the gradients are staying in the same direction, then the step size is too small. **Can we use this to make steps smaller when gradients reverse sign and larger when gradients are consistently in the same direction?**

Polyak momentum step. Adds an extra *momentum term* to gradient descent.

$$w_{t+1} = w_t - \alpha\nabla f(w_t) + \beta(w_t - w_{t-1}).$$

Intuition: if the current gradient is in the same direction as the previous step, move a little further in the same direction. If it's in the opposite direction, move a little less far. This is also known as the **heavy ball method** because it approximately simulates the dynamics of a ball that has physical momentum sliding through the space we want to optimize over.

Analysis of Polyak momentum. Rather than showing you the full analysis for convex functions (which is

involved), we'll look at a simple case that lets us build intuition: the case of one dimensional quadratics

$$f(w) = \frac{\lambda}{2} w^2.$$

Using this, Polyak momentum's update step looks like

$$\begin{aligned} w_{t+1} &= w_t - \alpha\lambda w_t + \beta(w_t - w_{t-1}) \\ &= (1 + \beta - \alpha\lambda) w_t - \beta w_{t-1}. \end{aligned}$$

Now we use a simplifying trick: let $w_t = \beta^{t/2} z_t$. Then

$$\beta^{(t+1)/2} z_{t+1} = \beta^{t/2}(1 + \beta - \alpha\lambda) z_t - \beta^{(t-1)/2} \cdot \beta z_{t-1}$$

which reduces to

$$z_{t+1} = \frac{1 + \beta - \alpha\lambda}{\sqrt{\beta}} z_t - z_{t-1}.$$

Next, if we let

$$u = \frac{1 + \beta - \alpha\lambda}{2\sqrt{\beta}},$$

then this simplifies further to

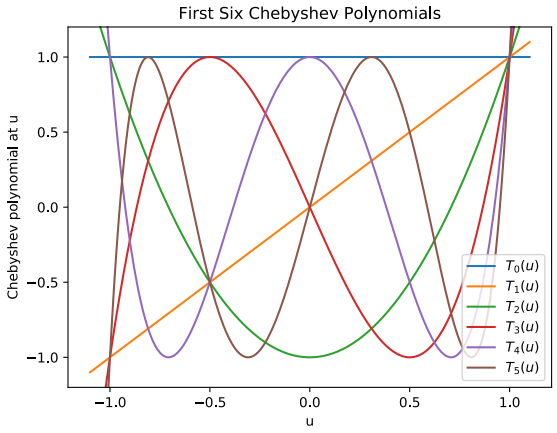
$$z_{t+1} = 2u z_t - z_{t-1}.$$

If we initialize with some fixed values of z_0 and z_1, this is a degree-t polynomial in u. If we choose to initialize with $z_0 = 1$ and $z_1 = u$ then these are a special family of polynomials called the **Chebyshev polynomials**, which are standardly defined by $T_0(u) = 1$, $T_1(u) = u$, and

$$T_{t+1}(u) = 2u T_t(u) - T_{t-1}(u).$$

These polynomials have an important and very useful property: for all $t \in \mathbb{N}$,

$$\text{if } -1 \le u \le 1 \text{ then } -1 \le T_t(u) \le 1.$$



This means that as long as $|u| \le 1$, it will also hold that $|z_t| \le 1$. And since we defined z_t in terms of w_t,

$$w_t^2 = \left(\beta^{t/2} z_t\right)^2 = \beta^t z_t^2 \le \beta^t$$

as long as we somehow arrange for $z_0 = 1$ and $z_1 = u$. **What if we initialize somewhere else?**

So w_t^2 will be approaching zero at a linear rate of β^t. Note that this happens no matter what the step size is, as long as $|u| \leq 1$. To make $|u| \leq 1$, we need

$$1 \geq |u| = \left| \frac{1 + \beta - \alpha\lambda}{2\sqrt{\beta}} \right| \quad \Rightarrow \quad -1 \leq \frac{1 + \beta - \alpha\lambda}{2\sqrt{\beta}} \leq 1.$$

What does this mean for multidimensional quadratics? For a multidimensional quadratic, we'll see the same dynamics as the single-dimension quadratic along each of the eigenvectors of the second derivative matrix. Derivation?

Importantly, they will all converge *at the same rate of* β^t, even for directions of different curvature! This wil happen as long as the condition

$$-1 \leq \frac{1 + \beta - \alpha\lambda}{2\sqrt{\beta}} \leq 1$$

is satisfied for *all* the directions λ. Since $\mu \leq \lambda \leq L$, this will hold if and only if

$$-1 \leq \frac{1 + \beta - \alpha L}{2\sqrt{\beta}} \quad \text{and} \quad \frac{1 + \beta - \alpha\mu}{2\sqrt{\beta}} \leq 1.$$

Now, we want to make β as small as possible to converge as fast as possible. How small can we make β for this to hold? This smallest value of β will result in these inequalities holding with equality, so we'll get

$$-1 = \frac{1 + \beta - \alpha L}{2\sqrt{\beta}} \quad \text{and} \quad \frac{1 + \beta - \alpha\mu}{2\sqrt{\beta}} = 1.$$

And through solving this for α and β (two equations, two unknowns), we get

$$\alpha = \frac{2 + 2\beta}{L + \mu} \quad \text{and} \quad \sqrt{\beta} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} = 1 - \frac{2}{\sqrt{\kappa} + 1}.$$

How does this compare to the rate that we got for gradient descent without momentum?

What is the additional computational cost of running momentum? What is the memory requirement?

How do we set the momentum for machine learning?

- Often, just set it to be $\beta = 0.9$
- Can also use a *hyperparameter optimization* method (which we'll cover later)

Nesterov momentum step. Slightly different from Polyak momentum; guaranteed to work for convex functions.

$$v_{t+1} = w_t - \alpha\nabla f(w_t)$$
$$w_{t+1} = v_{t+1} + \beta(v_{t+1} - v_t).$$

Main difference: separate the momentum state from the point that we are calculating the gradient at.

Momentum with SGD. Usually we run something like this:

$$v_{t+1} = \beta v_t - \alpha\nabla f_{i_t}(w_t)$$
$$w_{t+1} = w_t + v_{t+1}.$$