Lecture 8: 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

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

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

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

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 $L \ge \mu > 0$ 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

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

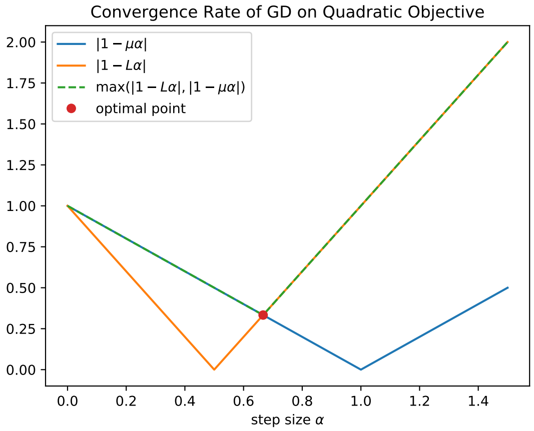
and so

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

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

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

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, we'd 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

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

We can write this in terms of a matrix multiplication as

$$\begin{bmatrix} w_{t+1} \\ w_t \end{bmatrix} = \begin{bmatrix} (1 + \beta - \alpha \lambda) & -\beta \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_t \\ w_{t-1} \end{bmatrix}.$$

This is sometimes called the *companion matrix* of the process. If we call this matrix M, then by induction we get

$$\begin{bmatrix} w_{t+1} \\ w_t \end{bmatrix} = M \begin{bmatrix} w_t \\ w_{t-1} \end{bmatrix} = M^2 \begin{bmatrix} w_{t-1} \\ w_{t-2} \end{bmatrix} = \cdots = M^t \begin{bmatrix} w_1 \\ w_0 \end{bmatrix}.$$

Now, we need to learn some things about M to make sense of this expression M^t. Assuming that M is diagonalizable (i.e. it has a full complement of eigenvectors) we can write the matrix in terms of its eigendecomposition as

$$M = Q \Lambda Q^{-1}$$

where Q is the matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues. Using this, we can write M^t as

$$M^t = \left(Q \Lambda Q^{-1} \right) \left(Q \Lambda Q^{-1} \right) \cdots \left(Q \Lambda Q^{-1} \right) = Q \Lambda^t Q^{-1}.$$

So the behavior of M^t is going to depend on the behavior of its eigenvalues. In particular,

$$\begin{aligned} w_{t+1}^2 \le \left\| \begin{bmatrix} w_{t+1} \\ w_t \end{bmatrix} \right\|^2 = \left\| M^t \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} \right\|^2 = \left\| Q \Lambda^t Q^{-1} \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} \right\|^2 &\le \|Q\|^2 \cdot \left\| \Lambda^t \right\|^2 \cdot \left\| Q^{-1} \right\|^2 \cdot \left\| \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} \right\|^2 \\ &= \|Q\|^2 \cdot \left\| Q^{-1} \right\|^2 \cdot \left\| \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} \right\|^2 \cdot \left(\max_{\chi \text{ an eigenvalue of } M} |\chi|^t \right)^2 \\ &= \|Q\|^2 \cdot \left\| Q^{-1} \right\|^2 \cdot \left\| \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} \right\|^2 \cdot \left(\max_{\chi \text{ an eigenvalue of } M} |\chi|^2 \right)^t \end{aligned}$$

So (barring issues with Q not being invertible) this algorithm is going to converge at a linear rate determined by the maximum absolute value of an eigenvalue of M. To understand how Polyak momentum converges **we need to understand the eigenvalues of M.**

To do this, first observe that the characteristic polynomial of M is

$$|\chi I - M| = \left| \begin{bmatrix} \chi - (1 + \beta - \alpha \lambda) & \beta \\ -1 & \chi \end{bmatrix} \right| = \chi \left(\chi - (1 + \beta - \alpha \lambda) \right) + \beta = \chi^2 - (1 + \beta - \alpha \lambda) \chi + \beta.$$

So, the eigenvalues of M are going to be

$$\chi = \frac{(1 + \beta - \alpha \lambda) \pm \sqrt{(1 + \beta - \alpha \lambda)^2 - 4 \beta}}{2}.$$

Now we make an important observation. If

$$(1 + \beta - \alpha \lambda)^2 - 4 \beta < 0, \quad \text{which is equivalent to} \quad -1 \le \frac{1 + \beta - \alpha \lambda}{2 \sqrt{\beta}} \le 1,$$

then the term in this square root will be negative. In this case, the eigenvalues are complex, and in particular they must be complex conjugates of each other. So they must have the same absolute value (because a complex number and its conjugate have the same absolute value) and the square of their absolute value must be equal to their product (because a complex number's absolute value is the square root of itself times its conjugate). Explicitly, if we call the eigenvalues χ_1 and χ_2, then

$$\chi_1^* = \chi_2 \qquad \text{and} \qquad |\chi_1|^2 = |\chi_2|^2 = \chi_1^* \chi_1 = \chi_1 \chi_2.$$

But this last term is just the product of the eigenvalues of M. And we know what the product of the eigenvalues of M is: that's just the determinant of M, which is β. So

$$|\chi_1| = |\chi_2| = \sqrt{\beta}.$$

This means that momentum converges at a rate of $\sqrt{\beta}$ in this case! Or, equivalently, $f(w_t) = \lambda w_t^2 / 2$ 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 our condition above is satisfied.

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 will 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

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

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

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

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

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

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.

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

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:

$$\begin{aligned} v_{t+1} &= \beta v_t - \alpha \nabla f_{\tilde{i}_t}(w_t) \\ w_{t+1} &= w_t + v_{t+1}. \end{aligned}$$