Accelerated First-Order Methods

Gradient descent (GD) and stochastic gradient descent (SGD) are known as first-order methods, since they make use of first-order derivatives (gradients) only. These tend to be computationally efficient but have some problems:

1) Not as fast as second-order methods (e.g., Newton's method), which also make use of the Hessian

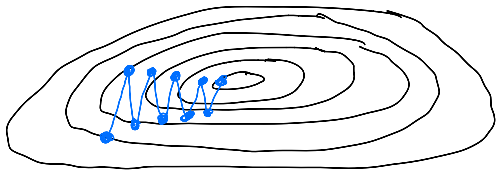
2) Choosing learning rate can be difficult

3) SGD may have high variance in its "descent" and in its performance

A variety of methods have been developed to address these issues. Namely, momentum methods address the first, adaptive methods the second, and mini-batch GD helps the third.

Momentum

One problem with SGD is that it can oscillate around the optimal point, making progress slow. For example, consider the two-dimensional problem below, where the concentric ellipses are contour lines, and the blue line indicates the path followed by SGD.



One way to handle this issue is by accounting for momentum in the movement, which "keeps the ball rolling" in the same direction as it's already moving. Suppose we wish to minimize

$$J(w) = \sum_{i=1}^{N} J_i(w)$$

The SGD update is then

$$V_k = \mu \nabla J_i(w_k)$$

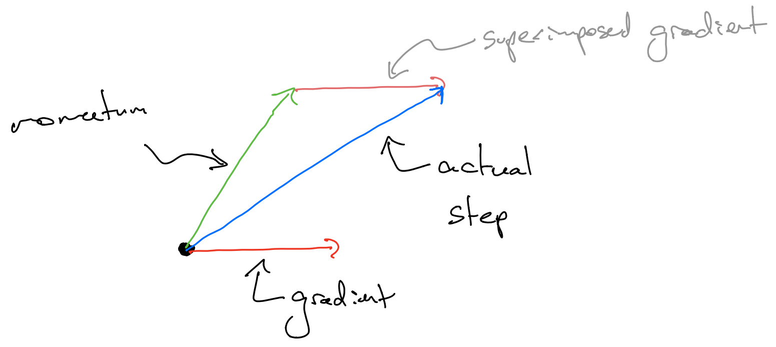
$$w_{k+1} = w_k - V_k$$

To incorporate momentum, we add a fraction of the previous update vector, yielding the update

$$V_k = \gamma V_{k-1} + \mu \nabla J_i(w_k) \quad \text{(momentum update)}$$

$$w_{k+1} = w_k - V_k$$

where γ is usually set to be around 0.9. The resulting step balances the momentum and gradient terms

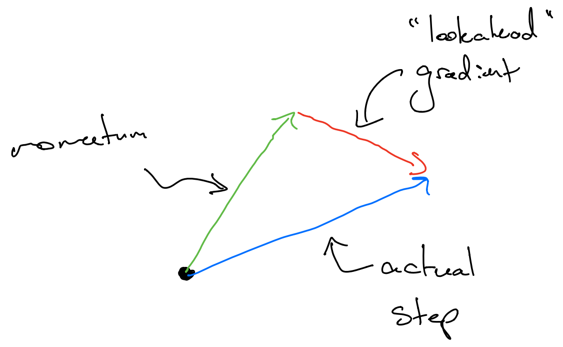


Momentum can be further improved by accounting for where the ball will go next, i.e., if we know a hill is coming that will turn us around, we may want to slow down in advance. This leads to a different update known as Nesterov accelerated gradient descent (NAG). The key idea is that we first think of where our momentum will take us and then compute the gradient from there. The update is

$$v_k = \gamma v_{k-1} + \mu \nabla J_i \big(\underbrace{w_k - \gamma v_{k-1}}_{\text{momentum step}} \big)$$

$$w_{k+1} = w_k - v_k$$

(Nesterov acceleration update)



The resulting step is faster and more stable in practice and can be shown to converge quadratically to the optimum, as opposed to linearly like GD.

Adaptivity

Momentum methods yield faster convergence by changing the step direction. Now we'll consider acceleration by changing the step size. One issue with this is that we may wish to take a different step size depending on the feature/dimension. For example, we may wish to have progress in each dimension even out over time. The first algorithm to handle this issue is known as Adagrad. Let

$$g_k^{(i)} = [\nabla J_i(w_k)]_i \in \mathbb{R}$$

be the ith coordinate of the gradient vector at step k. Adagrad weights the step size in this dimension by the accumulated gradient using the update

$$V_k^{(i)} = \mu \frac{g_k^{(i)}}{\sqrt{\underbrace{\sum_{\ell=1}^{k} g_\ell^{(i)2}}_{\text{previous gradients}} + \underbrace{\varepsilon}_{\text{regularization to avoid dividing by zero}}}}$$

$$w_k^{(i)} = w_{k-1}^{(i)} - V_k^{(i)}$$

where $\varepsilon \approx 10^{-6}$ avoids division by zero. The main drawback to Adagrad is that all weights are strictly decaying, which can lead to slow convergence. This issue is addressed by the Adadelta and RMSProp algorithms — we discuss the latter here since it's a bit more popular in deep learning.

Before proceeding, first note that we can write the vectorized Adagrad update as

$$v_k = \mu \frac{g_k}{\sqrt{\sum_{l=1}^{k} g_l^2 + \varepsilon}} \qquad \text{(Adagrad update)}$$

$$w_k = w_{k-1} - v_k$$

where $g_l^2 \in \mathbb{R}^D$ denotes the element-wise square of the D-dimensional gradient vector. The key idea behind RMSProp (and Adadelta) is to weight the step sizes by a moving window of previous gradients. Instead of maintaining this window explicitly, RMSProp defines the running average as

$$\bar{g}_0^2 = 0$$
$$\bar{g}_k^2 = \gamma \bar{g}_{k-1}^2 + (1-\gamma) g_k^2$$

where typically we set $\gamma = 0.9$. The RMSProp update is then

$$v_k = \mu \frac{g_k}{\sqrt{\bar{g}_k^2 + \varepsilon}} \qquad \text{(RMSProp update)}$$

$$w_k = w_{k-1} - v_k$$

Finally, methods such as Adam, AdaMax, and Nadam combine the ideas of momentum and adaptivity to further improve convergence speed.

Mini-Batch SGD

As mentioned above, SGD has drawbacks associated with its "stochastic" gradient update. Namely, it may not descend very well and its overall variance in finding the global optimum can be high - especially for non-convex problems like those associated with deep learning. Intuitively think of SGD as maintaining a noisy estimate of the average gradient

$$\bar{g}_k = \nabla J_i(w_k)$$

where $i \in 1, \dots, N$. This is like estimating the mean from a single sample, which we would never do in probability/estimation theory. In contrast, full GD sets

$$\bar{g}_k = \sum_{i=1}^{N} \nabla J_i(w_k),$$

which gives a low-noise estimate but at a high computational cost. A compromise between these two is mini-batch SGD, which sets

$$\bar{g}_k = \sum_{i \in B} \nabla J_i(w_k)$$

where $B \subset [1, \dots, N]$ is a subset of $M < N$ samples (drawn randomly). By taking M large enough, we reduce the noise in our gradient estimate, while by keeping $M \ll N$, we avoid the issues of computational complexity associated with batch GD. Typically, we take M between 50 - 256. All acceleration methods discussed above can be applied in the mini-batch case as well.