Momentum optimization

Imagine a bowling ball rolling down a gentle slope on a smooth surface: it will start out slowly, but it will quickly pick up momentum until it eventually reaches terminal velocity (if there is some friction or air resistance). This is the very simple idea behind *Momentum optimization*, proposed by Boris Polyak in 1964.[11] In contrast, regular Gradient Descent will simply take small regular steps down the slope, so it will take much more time to reach the bottom.

Recall that Gradient Descent simply updates the weights θ by directly subtracting the gradient of the cost function $J(\theta)$ with regards to the weights ($\nabla_\theta J(\theta)$) multiplied by the learning rate η. The equation is: $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$. It does not care about what the earlier gradients were. If the local gradient is tiny, it goes very slowly.

Momentum optimization cares a great deal about what previous gradients were: at each iteration, it adds the local gradient to the *momentum vector* **m** (multiplied by the learning rate η), and it updates the weights by simply subtracting this momentum vector (see Equation 11-4). In other words, the gradient is used as an acceleration, not as a speed. To simulate some sort of friction mechanism and prevent the momentum from growing too large, the algorithm introduces a new hyperparameter β, simply called the *momentum*, which must be set between 0 (high friction) and 1 (no friction). A typical momentum value is 0.9.

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta \mathbf{m} + \eta \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \mathbf{m}$

You can easily verify that if the gradient remains constant, the terminal velocity (i.e., the maximum size of the weight updates) is equal to that gradient multiplied by the learning rate η multiplied by $\frac{1}{1-\beta}$. For example, if $\beta = 0.9$, then the terminal velocity is equal to 10 times the gradient times the learning rate, so Momentum optimization ends up going 10 times faster than Gradient Descent! This allows Momentum optimization to escape from plateaus much faster than Gradient Descent. In particular, we saw in Chapter 4 that when the inputs have very different scales the cost function will look like an elongated bowl (see Figure 4-7). Gradient Descent goes down the steep slope quite fast, but then it takes a very long time to go down the valley. In contrast, Momentum optimization will roll down the bottom of the valley faster and faster until it reaches the bottom (the optimum). In deep neural networks that don't use Batch Normalization, the upper layers will often end up having inputs with very

11 "Some methods of speeding up the convergence of iteration methods," B. Polyak (1964).

different scales, so using Momentum optimization helps a lot. It can also help roll past local optima.

> Due to the momentum, the optimizer may overshoot a bit, then come back, overshoot again, and oscillate like this many times before stabilizing at the minimum. This is one of the reasons why it is good to have a bit of friction in the system: it gets rid of these oscillations and thus speeds up convergence.

Implementing Momentum optimization in TensorFlow is a no-brainer: just replace the `GradientDescentOptimizer` with the `MomentumOptimizer`, then lie back and profit!

```
optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9)
```

The one drawback of Momentum optimization is that it adds yet another hyperparameter to tune. However, the momentum value of 0.9 usually works well in practice and almost always goes faster than Gradient Descent.

Nesterov Accelerated Gradient

One small variant to Momentum optimization, proposed by Yurii Nesterov in 1983,[12] is almost always faster than vanilla Momentum optimization. The idea of *Nesterov Momentum optimization*, or *Nesterov Accelerated Gradient* (NAG), is to measure the gradient of the cost function not at the local position but slightly ahead in the direction of the momentum (see Equation 11-5). The only difference from vanilla Momentum optimization is that the gradient is measured at $\theta + \beta\mathbf{m}$ rather than at θ.

Equation 11-5. Nesterov Accelerated Gradient algorithm

$$1.\quad \mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta + \beta\mathbf{m})$$

$$2.\quad \theta \leftarrow \theta - \mathbf{m}$$

This small tweak works because in general the momentum vector will be pointing in the right direction (i.e., toward the optimum), so it will be slightly more accurate to use the gradient measured a bit farther in that direction rather than using the gradient at the original position, as you can see in Figure 11-6 (where ∇_1 represents the gradient of the cost function measured at the starting point θ, and ∇_2 represents the gradient at the point located at $\theta + \beta\mathbf{m}$). As you can see, the Nesterov update ends up

12 "A Method for Unconstrained Convex Minimization Problem with the Rate of Convergence $O(1/k^2)$," Yurii Nesterov (1983).