CS 189 Spring 2018 — Introduction to Machine Learning — Note 13

1 Gradient Descent Extensions

Recall gradient descent, an iterative first-order method that takes small steps opposite the direction of the gradient:

Algorithm 1: Gradient Descent

```
Initialize $\mathbf{w}^{(0)}$ to a random point
**while** $f(\mathbf{w}^{(t)})$ *not converged* **do**
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)})$
```

The standard gradient descent algorithm works fine in practice, but it can be significantly improved with some minor modifications. We now present some extensions that can greatly help with convergence and computational efficiency.

1.1 Gradient Descent with Momentum

Moving in the direction of steepest descent is a greedy approach — it uses information only about the current iterate without considering information about previous iterates or potential future iterates. This can often lead to oscillations that cause instability and slow convergence. These issues particularity arise when the objective function is disproportionately scaled — ie. the function is elongated along one axis while being contracted along along another, giving the illusion of "ravines" in the function landscape. The disproportionate scalings cause the algorithm to make large leaps in contracted directions, while making very slow progress along elongated directions. The resulting behavior is a series of oscillations that may reach the optimal point slowly or never reach it at all.

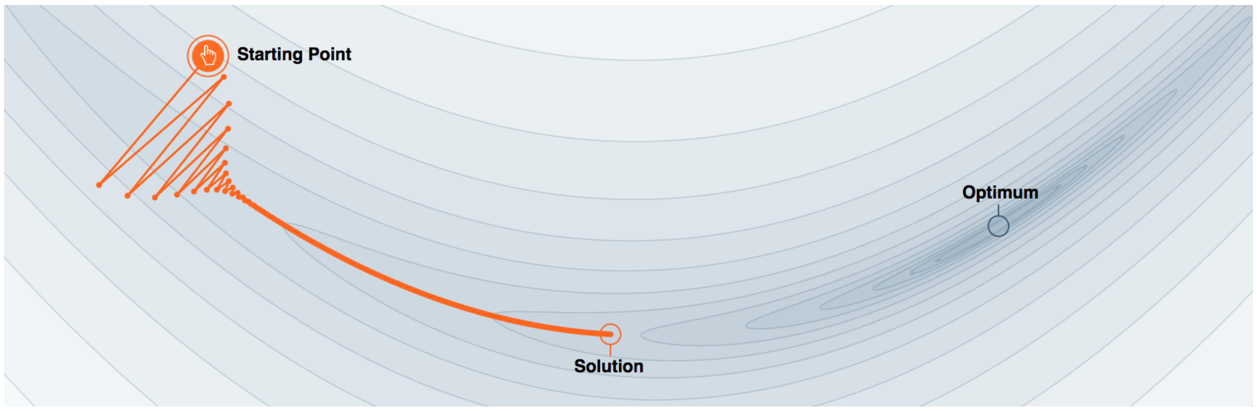

Figure 1: Standard gradient descent cannot converge to the optimum when the objective function is disproportionately scaled. Source: distill.pub

Polyak's heavy ball method addresses these issues by introducing a *momentum* term that adds inertia to the iterates and prevents them from deviating from the overall direction of the updates. Rather than updating the iterate $\mathbf{w}^{(t)}$ using the gradient $\nabla f(\mathbf{w}^{(t)})$, Polyak's heavy ball method uses $\nabla f(\mathbf{w}^{(t)})$ along with a history of all the gradients from the iterates seen so far. Specifically, it updates the iterates via a velocity term $\mathbf{v}^{(t)}$ that represents an exponential moving average of all of the gradients seen so far.

Algorithm 2: Polyak's Heavy Ball Method

```
Initialize $\mathbf{w}^{(0)}$ to a random point
Initialize $\mathbf{v}^{(0)}$ to $-\alpha_0 \nabla f(\mathbf{w}^{(0)})$
while $f(\mathbf{w}^{(t)})$ not converged do
    $\mathbf{v}^{(t)} \leftarrow \beta_t \mathbf{v}^{(t-1)} - \alpha_t \nabla f(\mathbf{w}^{(t)})$
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{v}^{(t)}$
```

The velocity term is updated in the following recursive fashion:

$$\mathbf{v}^{(t)} \leftarrow \beta_t \mathbf{v}^{(t-1)} - \alpha_t \nabla f(\mathbf{w}^{(t)})$$

which when unrolled, is equivalent to

$$\begin{aligned}\mathbf{v}^{(t)} \leftarrow & -\beta_t \cdot \beta_{t-1} \dots \beta_1 \alpha_0 \nabla f(\mathbf{w}^{(0)}) \\ & -\beta_t \cdot \beta_{t-1} \dots \beta_2 \alpha_1 \nabla f(\mathbf{w}^{(1)}) \\ & \dots \\ & -\alpha_t \nabla f(\mathbf{w}^{(t)})\end{aligned}$$

which in the case when the β's and α's are constant is equivalent to

$$\mathbf{v}^{(t)} \leftarrow -\beta^t \alpha \nabla f(\mathbf{w}^{(0)}) - \beta^{t-1} \alpha \nabla f(\mathbf{w}^{(1)}) - \dots - \alpha \nabla f(\mathbf{w}^{(t)})$$

The motivation for using a moving average of the gradients is as follows: we want to downplay the directions for which the gradient is oscillating over time and boost the directions for which the gradient is constant over time. When we are in a "ravine" this has the effect of "killing" the gradient in constricted directions whose derivatives oscillate over time, accelerating convergence.

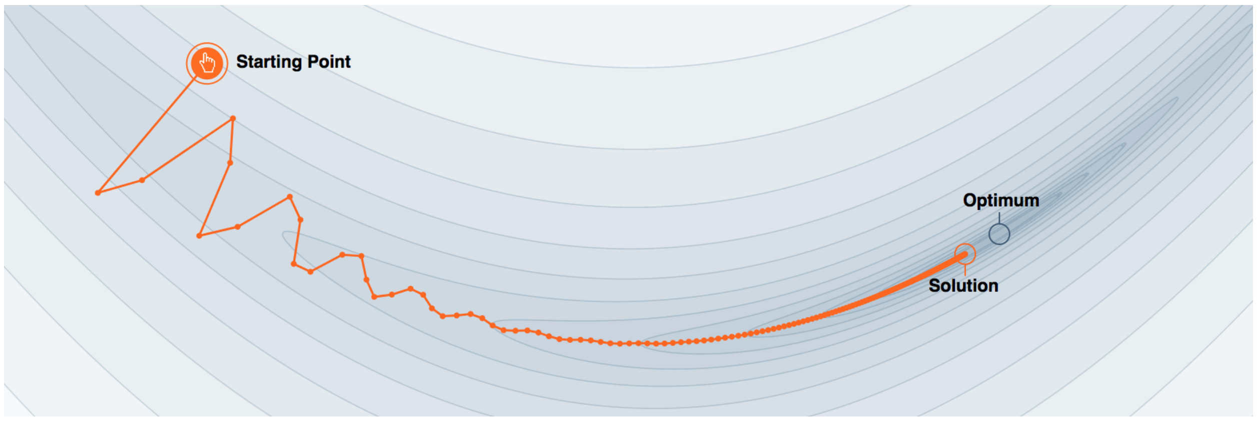

Figure 2: Polyak's heavy ball method uses momentum to dampen oscillations, accelerating convergence to the optimum point. Source: distill.pub

There is an alternative interpretation of Polyak's heavy ball method that is condensed to just one line:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) + \beta_t(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})$$

We can establish equivalence through the following manipulations:

$$\begin{aligned}
\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} + \mathbf{v}^{(t)} \\
&= \mathbf{w}^{(t)} + \beta_t \mathbf{v}^{(t-1)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) \\
&= \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) + \beta_t \mathbf{v}^{(t-1)} \\
&= \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) + \beta_t(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})
\end{aligned}$$

Polyak's heavy ball method uses information about past iterates to determine the descent direction. **Nesterov's accelerated gradient descent** improves on this reasoning, incorporating information about potential future iterates as well. The only difference in Nesterov's accelerated gradient descent is that it computes a "lookahead gradient" $\nabla f(\mathbf{w}^{(t)} + \beta_t \mathbf{v}^{(t-1)})$ instead of the gradient at the current iterate $\nabla f(\mathbf{w}^{(t)})$. Effectively, we are performing a one step "look ahead" of the gradient and moving in that direction, potentially correcting for oscillations ahead of us.

Algorithm 3: Nesterov's Accelerated Gradient Descent

```
Initialize $\mathbf{w}^{(0)}$ to a random point
Initialize $\mathbf{v}^{(0)}$ to $-\alpha_0 \nabla f(\mathbf{w}^{(0)})$
while $f(\mathbf{w}^{(t)})$ not converged do
    $\mathbf{v}^{(t)} \leftarrow \beta_t \mathbf{v}^{(t-1)} - \alpha_t \nabla f(\mathbf{w}^{(t)} + \beta_t \mathbf{v}^{(t-1)})$
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{v}^{(t)}$
```

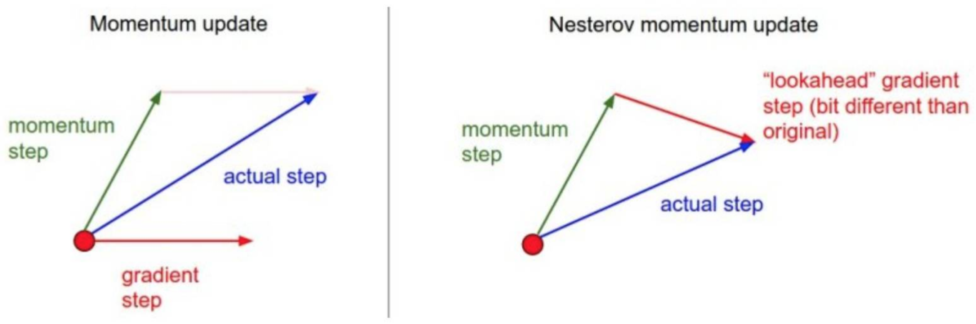

Figure 3: Polyak's heavy ball method applies gradient before update, while Nesterov's accelerated gradient descent applies gradient after update. Source: Stanford CS 231n

Through the same manipulations that we showed for Polyak's heavy ball method, we derive the one-line update for Nesterov's accelerated gradient descent:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)} + \beta(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})) + \beta_t(\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)})$$

1.2 Stochastic Gradient Descent

In the standard gradient decent update, computing the gradient $\nabla f(\mathbf{w}^{(t)})$ can be an expensive operation. Imagine an objective function of the form

$$f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w})$$

where $n \gg d$. Computing the gradient effectively entails computing and adding n separate gradients, which is very costly:

$$\nabla f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{w})$$

Stochastic gradient descent aims to resolve this issue by replacing the gradient with a noisy **stochastic gradient** $G(\mathbf{w})$ that is significantly more efficient to compute. Specifically, the stochastic gradient is a random vector that is an *unbiased estimate* of the true gradient:

$$\mathbb{E}[G(\mathbf{w})] = \nabla f(\mathbf{w})$$

The stochastic gradient is used in place of the true gradient in the update rule:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla G(\mathbf{w}^{(t)})$$

Stochastic gradient descent is mainly used when the objective is a sum of decomposable, independent and identically distributed (i.i.d) losses $f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w})$. This naturally entails the stochastic gradient

$$G(\mathbf{w}) = \nabla f_i(\mathbf{w})$$

by just drawing an index i uniformly at random from $\{1, \ldots, n\}$. Since the losses are i.i.d, we have that

$$\mathbb{E}[G(\mathbf{w})] = \mathbb{E}[\nabla f_i(\mathbf{w})] = \nabla \mathbb{E}[f_i(\mathbf{w})] = \nabla f(\mathbf{w})$$

Algorithm 4: Stochastic Gradient Descent

```
Initialize $\mathbf{w}^{(0)}$ to a random point
while $f(\mathbf{w}^{(t)})$ not converged do
    Sample a random index $i_t$
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f_{i_t}(\mathbf{w}^{(t)})$
```

Now, we just have to perform one gradient computation in each update step rather than n separate gradient computations, leading each iteration to become significantly faster. However, since our stochastic gradients comprise of just a single sample, they may have very large variance, causing the performance of the algorithm to oscillate quite frequently and requiring us to perform a significantly higher number of iterations.

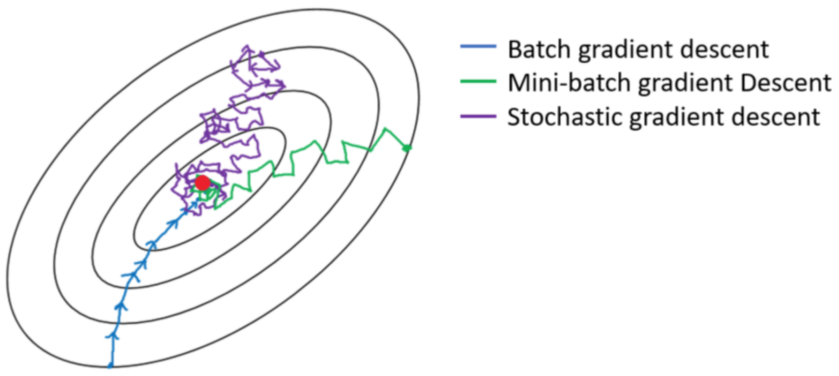

Figure 4: Increasing the batch size will lead to more stability at the cost of higher computational costs. Source: Towards Data Science