

Lec 11: Newton's Method, Momentum and Stochastic Gradient Descent (CS3780/5780, Sp26)

1 Quick Recap

We considered an optimization problem of the form

$$\text{Minimize}_{\mathbf{w} \in \mathbb{R}^d} \ell(\mathbf{w})$$

The gradient descent algorithm started with some initial guess \mathbf{w}_0 and updated this guess iteratively using the update rule

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla \ell(\mathbf{w}_t)$$

where $\eta > 0$ is referred to as step-size. The intuition behind the update rule stemmed from a first order approximation of a function locally around a point of form

$$\ell(\mathbf{w} + \mathbf{s}) \approx \ell(\mathbf{w}) + \mathbf{s}^\top \nabla \ell(\mathbf{w}) + O(\|\mathbf{s}\|^2)$$

which gives us good linear approximation of a function for small \mathbf{s} . Given this, it made sense for us to find an \mathbf{s} that minimized the right hand side above which yielded

$$\mathbf{s} = -\eta \nabla \ell(\mathbf{w})$$

which ensured that every step of gradient descent improved the objective (until we hit a flat region at which point we converge). We also covered Adagrad that could be seen as having different step sizes for the different coordinates with these step sizes depending inversely on square-root of sum of past gradient squares on the corresponding coordinate. The basic intuition was if gradients in a coordinate are large we prefer smaller step sizes and vice versa.

2 Newton's Method

To build the intuition for gradient descent we approximated $\ell(\mathbf{w} + \mathbf{s})$ by a linear function of \mathbf{s} around \mathbf{w} using first order Taylor expansion. Instead one can use second order Taylor approximation to obtain a sharper approximation of $\ell(\mathbf{w} + \mathbf{s})$ around \mathbf{w} in terms of a quadratic in \mathbf{s} . In the one dimensional case this equation looks like

$$\ell(w + s) \approx \ell(w) + s \cdot \ell'(w) + \frac{1}{2} \ell''(w) s^2$$

The idea behind Newton's method is to simply optimize this approximation w.r.t. s to get

$$s_t = \arg \min s \ell'(w) + \frac{1}{2} \ell''(w_t) s^2 = -\frac{1}{\ell''(w_t)} \ell'(w_t)$$

For the d -dimensional case the second order approximation becomes

$$\ell(\mathbf{w} + \mathbf{s}) \approx \ell(\mathbf{w}) + \mathbf{s}^\top \nabla \ell(\mathbf{w}) + \frac{1}{2} \mathbf{s}^\top \nabla^2 \ell(\mathbf{w}) \mathbf{s}$$

where $\nabla^2 \ell(\mathbf{w})$ is a $d \times d$ square matrix called the Hessian matrix with $\nabla^2 \ell(\mathbf{w})[i, j] = \frac{\partial^2}{\partial \mathbf{w}[i] \partial \mathbf{w}[j]} \ell(\mathbf{w})$ and $\nabla^2 \ell(\mathbf{w})[i, i] = \frac{\partial^2}{\partial \mathbf{w}[i]^2} \ell(\mathbf{w})$. Again optimizing over \mathbf{s} around \mathbf{w}_t yields Newton's method in higher dimension as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(\nabla^2 \ell(\mathbf{w}_t) \right)^{-1} \nabla \ell(\mathbf{w}_t)$$

Since the second order approximation only holds for small \mathbf{s} , this Newton's steps in general is valid only when we are close enough to the minima so that the second order approximation makes sense.

In the one dimensional case, Newton's method can be seen as simply choosing adaptively stepsize as $-1/\ell''(w_t)$ but in the multi-dimensional case, the $(\nabla^2 \ell(\mathbf{w}_t))^{-1}$ is more than just a step-size and affect the direction of step taken. Newton step can be viewed as *preconditioned* gradient descent with a matrix that adapts to local curvature given by this so called Hessian matrix. In directions of high curvature, $(\nabla^2 \ell(\mathbf{w}_t))^{-1}$ takes smaller steps; in flat directions, it takes larger steps.

2.1 Newton's Method is a One Step Optimizer for Quadratic Convex Loss

Say $\ell(\mathbf{w}) = \frac{1}{2} \mathbf{w}^\top A \mathbf{w} + \mathbf{b}^\top \mathbf{w}$ for some positive definite matrix A . The claim is that one step of Newton's method starting from any \mathbf{w}_0 goes directly to the minimizer of the loss. Do you see this? Why is this the case?

Well at an intuitive level this is obvious, Newton's method optimizes the second order Taylor approximation of a function, but for a quadratic function, the second order Taylor approximation is not an approximation but an equality and so Newton's step is simply a one step minimization. More succinctly:

$$\mathbf{w}_1 = \mathbf{w}_0 - \left(\nabla^2 \ell(\mathbf{w}_0) \right)^{-1} \nabla \ell(\mathbf{w}_0)$$

But note that

$$\nabla^2 \ell(\mathbf{w}_0) = A$$

and $\nabla \ell(\mathbf{w}_0) = A \mathbf{w}_0 + \mathbf{b}$ and so

$$\mathbf{w}_1 = \mathbf{w}_0 - A^{-1}(A \mathbf{w}_0 + \mathbf{b}) = -A^{-1} \mathbf{b}$$

But this is exactly the minimizer of $\ell(\mathbf{w})$.

Damped Newton's Method Newton's method directly optimizes the second order approximation which only makes sense when we are close to minima. A trick often used to improve Newton's method is to do the following update instead

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \left(\nabla^2 \ell(\mathbf{w}_t) \right)^{-1} \nabla \ell(\mathbf{w}_t)$$

where one every round α is obtained by looking for the largest $\alpha \leq 1$ satisfying the equation

$$\ell(\mathbf{w}_t - \alpha \left(\nabla^2 \ell(\mathbf{w}_t) \right)^{-1} \nabla \ell(\mathbf{w}_t)) < \ell(\mathbf{w}_t) - \frac{1}{2} \alpha \left(\nabla^2 \ell(\mathbf{w}_t) \right)^{-1} \nabla \ell(\mathbf{w}_t)$$

Words of Wisdom:

1. Newton's method not only requires computing a $d \times d$ Hessian matrix but also inverting it at every steps which is of order $O(d^{2.8})$ time complexity (practically $O(d^3)$)
2. For Newton's method to work well we need to be not too far from the minima. Hence typically one starts with few steps of gradient descent as warm start.
3. Newton's method can have stability issues especially when the Hessian matrix is ill conditioned (barely invertible)
4. One uses Newton's method when d is not too large and when one wants high precision.
5. For strongly convex loss ℓ one can get provable fast convergence for Newton's method (and Hessian in this case is positive definite by definition)

3 Momentum

For modern large scale machine learning problem dimension d is very large and methods where each iteration costs anything more than linear time in d can be prohibitive. Hence methods like gradient descent have been more popular.

We will now look at two different issues of gradient descent algorithm and will come up with a single fix for both these issues. For the **first**, Imagine using gradient descent to move down to the minimum of a sharp valley. We can choose super small step-size but then we would move very slowly to the minima. Alternatively we could pick a reasonable large step-size, but in this case we would bounce around the minima several times before finally converging or getting close to the bottom. To see this you can even consider our favorite 1D example $\ell(w) = w^2$. If $\eta = 0.0005$ and $w_0 = 1$ we would take about a thousand steps to get to the minima but if $\eta = 0.75$ say then we would oscillate to the left and right of 0 before we finally get closer to minima as well.

As for the **second** issue, consider a function like

$$\ell(w) = w^4 - \frac{1}{2}w^2 + \frac{1}{10}w$$

This function has a global minima to the left of 0 and a smaller local minima to the right of 0. Now say we started our optimization at $w_0 = 1$. A common failure mode of gradient descent in such a scenario would be for gradient descent to get stuck at the first sub-optimal local minima and not move since gradient is 0 or very small near this minima. But is this dip is indeed small think about our analogy of skiing, what would happen, our skier would have momentum and their speed coming down the slope would give them enough inertia to go over the small bump and proceed to the big valley (global minima).

This idea of momentum can be added to gradient descent and can address both the above issues. The gradient descent algorithm with momentum added in is given by

Initialize $\mathbf{w}_0 \in \mathbb{R}^d$, $\mathbf{v}_0 = 0$, $t = 0$, $\eta > 0$, $\gamma \in (0, 1)$ Converged = False

While (Converged == False):

$$\mathbf{v}_{t+1} = \gamma \mathbf{v}_t - (1 - \gamma) \nabla \ell(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{v}_{t+1}$$

$$t = t + 1$$

If $\|\mathbf{w}_t - \mathbf{w}_{t-1}\| < \delta$ Then Converged = True

End While

The basic idea is that we replace the gradient by $1 - \gamma$ times the current gradient and γ times a combination of past gradients. Why does this help? Think about the first issue, in this case, in the first step yes we jump from right of the minima to the left. But now when taking the next gradient step with momentum, our gradient pushes us back to the right of the minima by a bunch, but this push is dampened by the mixing in of the gradient from the previous step which had pushed us to the left in the first place. Hence, overall in step 2 the push to the right is smaller and as a result our jumping around the minima is much more smooth and we settle to the bottom quicker. Similarly consider effect of momentum on the second issue mentioned above. We travel down our slope and reach a flat (or slightly upward inclined) portion of our function. In this case, gradient would be a very small value (or might even push us back). But then our momentum term which carries past gradients will take us past this flat region or even the slight bump as long as the region is not too large. Overall momentum pushes us faster if we are traveling down a slope and smoothens our oscillation near minima. Overall, the main qualitative role of momentum:

- it smooths updates by aggregating gradients over time,
- it can preserve motion through flat regions where gradients are small, and
- it can help cross shallow barriers that would trap plain gradient descent in nearby local minima.

4 Stochastic Gradient Descent (SGD)

Recall that our main motivation to study these optimization techniques were to learn machine learning models that were solutions to optimization problems of form

$$\arg \min_{\mathbf{w} \in \mathbb{R}^d} \ell(\mathbf{w}) = \arg \min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \ell_i(\mathbf{w})$$

In modern ML settings, typically number of samples n and number of parameters d are both very large. So far we simply say $\ell(\mathbf{w})$ as some arbitrary function of \mathbf{w} , we didn't specifically use the fact that its the average loss function over n points. Note that if computing the gradient $\ell_i(\mathbf{w})$ took time order $O(d)$ (in general cant take lesser since its a d dimensional vector). Then generally, computing

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

would take time $O(nd)$ since we need to compute n gradients and average them. This could be expensive.

However, especially when n is large, one can use a neat trick to get a much faster algorithm. We can think of $\nabla\ell(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \nabla\ell_i(\mathbf{w})$, the average gradient as the expected gradient where the distribution w.r.t. which we take the expectation is simply the uniform distribution over gradients $\nabla\ell_1(\mathbf{w}), \dots, \nabla\ell_n(\mathbf{w})$. What is a good approximation for an expectation?

Well, taking average over samples drawn from the distribution! This suggests that one could approximate each gradient step by replacing the gradient $\nabla\ell(\mathbf{w}_t)$ by an average over a sampling of b gradients from this distribution. That is, we randomly choose b indices in $\{1, \dots, n\}$, say r_1, \dots, r_b and we perform the update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \frac{1}{b} \sum_{j=1}^b \nabla\ell_{r_j}(\mathbf{w}_t)$$

with the idea that $\frac{1}{b} \sum_{j=1}^b \nabla\ell_{r_j}(\mathbf{w}_t)$ is an approximation for gradient $\nabla\ell(\mathbf{w}_t)$. This way, per round of gradient descent, the gradient computation is only $O(bd)$ where b is referred to as the minibatch size.

Here is the really surprising part, this algorithm works great even if $b = 1$! At first this seems very surprising because it seems like taking one sample to get an approximation for an expectation is ridiculous. But a more careful analysis tells us why this approach works, while it is true that we are getting very very noisy (albeit unbiased) sample of gradient per round, since we take multiple steps of SGD, the noise over the multiple rounds cancel out. So net effect, with SGD, if we take many iterations/updates steps, the algorithm works. Of course this also suggests that compared to exact GD for SGD we would need many more updates. However, the fact that cost of each update is of order $O(bd)$ rather than $O(nd)$ comes to save us. In fact, for convex problems one can show that one single pass over the n data points using SGD (i.e. n updates with $b = 1$) produces result comparable with running GD to convergence. That is, in the time it costs to do one update of full GD, SGD produces a near optimal solution!

SGD algorithm pseudo-code is as follows:

Initialize $\mathbf{w}_0 \in \mathbb{R}^d$, $\eta > 0$, mini-batch size $b \in [n]$

For $t = 1$ to T

 Pick mini-batch $r_1, \dots, r_b \in [n]$ (Eg. sample or cycle over data)

 Compute $\nabla_t = \frac{1}{b} \sum_{i=1}^b \nabla\ell_{r_i}(\mathbf{w}_t)$

 Update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_t$

End For

A common way to pick the mini-batches is to first simply shuffle all the data and then in each update step just pick the subsequent b points as minibatch in that order and cycle through data as and when we have exhausted points.

4.1 Why Does SGD Work

While a more accurate analysis is more involved, a simple computation should convince us as to why SGD should work. Say at a given point \mathbf{w}_t , we have that ∇_t is an estimate of $\nabla\ell(\mathbf{w}_t)$ such that $\mathbb{E}[\nabla_t] = \nabla\ell(\mathbf{w}_t)$. From our first order Taylor theorem we have

$$\ell(\mathbf{w}_{t+1}) - \ell(\mathbf{w}_t) \leq -\eta \nabla\ell(\mathbf{w}_t)^\top \nabla_t + \frac{H}{2} \|\nabla_t\|^2$$

Taking expectation over the sampling of minibatch for that time we get

$$\begin{aligned}
\mathbb{E}[\ell(\mathbf{w}_{t+1}) - \ell(\mathbf{w}_t)] &\leq \mathbb{E}\left[-\eta \nabla \ell(\mathbf{w}_t)^\top \nabla_t + \frac{H\eta^2}{2} \|\nabla_t\|^2\right] \\
&= -\eta \mathbb{E}\left[\nabla \ell(\mathbf{w}_t)^\top \mathbb{E}_t[\nabla_t]\right] + \frac{H\eta^2}{2} \mathbb{E}\left[\|\nabla_t\|^2\right] \\
&= -\eta \mathbb{E}\left[\|\nabla \ell(\mathbf{w}_t)\|^2\right] + \frac{H\eta^2}{2} \mathbb{E}\left[\|\nabla_t\|^2\right]
\end{aligned}$$

Rearranging we get

$$\mathbb{E}\left[\|\nabla \ell(\mathbf{w}_t)\|^2\right] \leq \frac{\mathbb{E}[\ell(\mathbf{w}_t) - \ell(\mathbf{w}_{t+1})]}{\eta} + \frac{H\eta}{2} \mathbb{E}\left[\|\nabla_t\|^2\right]$$

Averaging over t from 1 to T we get

$$\begin{aligned}
\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla \ell(\mathbf{w}_t)\|^2\right] &\leq \frac{1}{\eta T} \sum_{t=0}^{T-1} (\mathbb{E}[\ell(\mathbf{w}_t) - \ell(\mathbf{w}_{t+1})]) + \frac{H\eta}{2} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla_t\|^2\right] \\
&= \frac{1}{\eta T} (\mathbb{E}[\ell(\mathbf{w}_0) - \ell(\mathbf{w}_T)]) + \frac{H\eta}{2} \frac{1}{T} \sum_{t=1}^T \mathbb{E}\left[\|\nabla_t\|^2\right]
\end{aligned}$$

Say the loss function was non-negative and say that the gradients ∇_t are such that $\|\nabla_t\| \leq B$ then we have:

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla \ell(\mathbf{w}_t)\|^2\right] \leq \frac{1}{\eta T} \ell(\mathbf{w}_0) + \frac{H\eta}{2} B^2$$

Thus we see that if we set η to be small, say to be line $1/\sqrt{T}$ then we see that after T steps of SGD the average expected norm-square of the gradients along the way become as small as order $1/\sqrt{T}$.

Hence we see that SGD gets to a region where the average gradients are small and if we for instance assume convexity or that the function has only global minima this above statement can be used to further conclude that SGD will converge to the optimum.