10-707: Advanced Deep Learning **Spring 2020**

Lecture 5: January 29

Lecturer: Andrej Risteski *Scribes: Michael Huang,Michelle Ma,Anish Sevekari,Yiwei Lyu*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

This lecture's notes illustrate some uses of various LaTeX macros. Take a look at this and imitate.

5.1 Supervised Learning and Optimization

In supervised learning, the empirical risk minimization approach focuses on minimizing a training loss over some class of functions F. In this context, there are three important questions:

1. How expressive is the class? Representational Power. In previous lectures, we explored this by looking at the universal approximation theorem, and exploring the applications of networks in convolutional architectures.

2. How do we minimize the training loss effectively? Optimization. What this lecture is focused on

3. How does the model perform on unseen samples

Typically, a training task in ml can be cast as minimizing some function f(x). In practice, most algorithms to find the minimum are iterative. It is cheap to calculate f(x), $\nabla f(x)$, but expensive to calculate higher order derivatives.

5.1.1 Gradient Descent

The simplest optimization, gradient descent: Taylor expand and move int he direction of steepest descent.

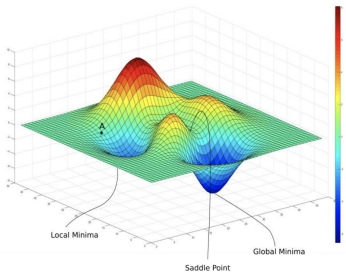
By Taylor's theorem, we have $f(x+\Delta) \approx f(x) + \Delta^T \nabla f(x) + O(||\Delta||^2)$

So, if we ignore higher-order effects, we have

$$\underset{\Delta, ||\Delta|| \leq \epsilon}{\operatorname{argmin}} \{ f(x+\Delta) - f(x) \} = -\epsilon \frac{\nabla f(x)}{||\nabla f(x)||}$$

After gradient descent, we will arrive at a point where $\nabla f(\hat{x}) \approx 0$ i.e we stop moving at a "stationary point".

5.1.2 Types of Stationary Points



Global minimum: actual minimizer, namely $f(\hat{x}) \leq f(x), \forall x \in \mathbb{R}^d$

Local minimum: $f(\hat{x}) \leq f(x),\ \forall x \text{ s.t. } ||x - \hat{x}|| \leq \epsilon$ for some $\epsilon > 0$

Local maximum: $f(\hat{x}) \geq f(x),\ \forall x \text{ s.t. } ||x - \hat{x}|| \leq \epsilon$ for some $\epsilon > 0$

Saddle points: stationary point that is *not* a local min/max.

Global Minimum: Finding these is very hard (NP-Hard) in practice.
Local Minimum: Easier to find than global minimum, and even work quite well, even though theoretically the difference between a local min and a global min can be very large.
Saddle Point: Usually undesirable, and arise from invariances in the input.

To determine what type of stationary point a point is, we can use second order checks:

Taylor's thm: $f(x + \Delta) \approx f(x) + \Delta^T \nabla f(x) + \frac{1}{2}\Delta^T \nabla^2 f(x)\Delta + O(||\Delta||^3)$

$$\approx f(x) + \frac{1}{2}\Delta^T \nabla^2 f(x)\Delta + O(||\Delta||^3)$$

If $\nabla^2 f(x) \succ 0$: for any direction Δ, and small enough $||\Delta||$

$\Delta^T \nabla^2 f(x)\Delta + O(||\Delta||^3) \geq 0$, so $f(x + \Delta) > f(x)$

Local minimum! (Flipped for local maximum)

If $\nabla^2 f(x)$ has both positive and negative eigenvalues:

Saddle point (not a local minimum/maximum)

If neither of these attains, test is inconclusive!

5.2 The Descent Lemma

Regarding the step size Δ in gradient descent, if $\|\Delta\|$ is too large, the algorithm can skip over local minima, and if $\|\Delta\|$ is too small, the algorithm will be too slow. In order to find the "sweet spot", we can use the descent lemma.

Lemma 5.1 *Let f be twice differentiable and $\|\nabla^2 f(x)\|_2 \leq \beta$. Then, setting $\eta = \frac{1}{\beta}$, and calling x_t the iterates of gradient descent, namely $x_{t_1} = x_t - \eta \nabla f(x_t)$, we have that:*

$$f(x_t) - f(x_{t+1}) \geq \frac{1}{2\beta}\|\nabla f(x_t)\|_2^2$$

In other words, the descent lemma lower bounds the difference between subsequent steps of gradient descent.

Use of Descent Lemma: Suppose f is lower bounded (e.g. $f \geq 0$), and $f(x_0) \leq M$ where M is a constant. Suppose we want point x_t s.t. $||\nabla f(x_t)|| \geq \epsilon$.
We now introduce **Lyapunov (potential) fn argument:** Suppose $\forall t \in [0, T], ||\nabla f(x_t)|| \geq \epsilon$. Then $f(x_T) \leq f(x_0) - T\frac{1}{2\beta}\epsilon \leq M - T\frac{1}{2\beta}\epsilon$.
Since we also know that we can lower bound f such that $f(x_T) \geq 0$, we then know that $M - T\frac{1}{2\beta}\epsilon \geq 0$, which gives us $T \leq 2M\beta/\epsilon$

Proof: By Taylor expansion and the mean value theorem, we have:

$$f(x + \Delta) = f(x) + \Delta^T \nabla f(x) + \frac{1}{2}\Delta^T \nabla^2 f(y)\Delta$$

Moreover, $\Delta^T \nabla^2 f(y)\Delta \leq ||\nabla^2 f(y)||_2 ||\nabla||_2^2 \leq \beta||\Delta||_2^2$ by rearranging and algebra, as well as the assumption made in the descent lemma. Plugging in $\Delta = -\eta\nabla f(x_t)$ (the step size we take during gradient descent):

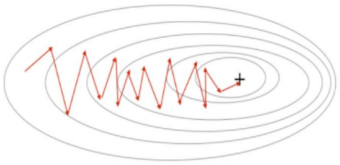
$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) - \eta||\nabla f(x_t)||^2 + \frac{1}{2}\beta\eta^2||\nabla f(x_t)||^2 \\ &= f(x_t) - \frac{1}{\beta}||\nabla f(x_t)||^2 + \frac{1}{2\beta}||\nabla f(x_t)||^2 && \text{(Plugging in } \eta = \tfrac{1}{\beta}\text{)} \\ &= f(x_t) - \frac{1}{2\beta}||\nabla f(x_t)||^2. \end{aligned}$$

Thus we've proved that $f(x_t) - f(x_{t+1}) \geq \frac{1}{2\beta}||\nabla f(x_t)||^2$ ■

5.3 Understanding Gradient Descent Locally

Let's restrict our attention to functions f which are quadratics. We can always approximate any given function by a quadratic near a local minima using taylor's theorem after ignoring third order terms. We want to study which quadratics are good and which are bad for gradient descent. Intuitively, convergence rate of gradient descent is low when the gradients don't directly point towards the minimizer, because this will cause the path taken by gradient descent to zig-zag a lot instead of moving in a straight line towards the minimizer.

Intuition: Notice that gradients are perpendicular to the level sets of the function, and therefore, if we have spherical contours (level sets) then gradient will point roughly towards the minimizer, giving us fast convergence. So we expect quadratics with ellipsoidal contours to perform much worse than those with spherical contours. This behavior is demonstrated in the figure on the right. The precise question is formulated below, and its answer is given by theorem 5.2.



Question: Let $f(x) = \frac{1}{2}x^\intercal Ax$. Can we characterize convergence time of gradient descent more precisely? What does it depend on?

Theorem 5.2 *Let A be a symmetric positive-definite matrix with minimum and maximum eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$ respectively. Let*

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$

denote the ***condition number*** *of the matrix. Then the iterates of gradient descent with rate* $\eta = \frac{2}{\lambda_{\max}+\lambda_{\min}}$ *satisfy:*

$$\|x_t\| \leq \left(\frac{\kappa-1}{\kappa+1}\right)^t \|x_0\|$$

Note that the minimizer for $f = \frac{1}{2}x^\intercal Ax$ is 0 since A is positive-definite. Therefore, $\|x_t\|$ is the distance of the t^{th} iterate of the gradient descent from the optimum. Also, observe that

$$\frac{\kappa-1}{\kappa+1} = 1 - \frac{2}{\kappa+1}$$

goes towards 1 as κ grows. Therefore, larger κ means lower convergence rate. Larger values of κ roughly correspond to how ellipsoidal the level curves of f are, so this theorem captures the intuitive reasoning above.

Proof: The update of gradient descent is given by $x_{t+1} = x_t - \eta\nabla f(x_t)$. And $\nabla f(x) = \nabla(\frac{1}{2}x^\intercal Ax) = Ax$. Therefore, we have

$$\begin{aligned}
\|x_{t+1}\| &= \|x_t - \eta\nabla f(x_t)\| \\
&= \|x_t - \eta Ax_t\| && \text{Since } \nabla f = Ax \\
&= \|(I-\eta A)x_t\| \\
&\leq \|I-\eta A\|_2\|x_t\| \\
&\leq \max(|1-\eta\lambda_{\max}|, |1-\eta\lambda_{\min}|)\|x_t\| \\
&= \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}}\|x_t\| && (5.1) \\
&= \frac{\kappa-1}{\kappa+1}\|x_t\|
\end{aligned}$$

The equation (5.1) holds since

$$|1-\eta\lambda_{\max}| = \left|1 - \frac{2\lambda_{\max}}{\lambda_{\max}+\lambda_{min}}\right| = \frac{|\lambda_{\min}-\lambda_{\max}|}{\lambda_{\max}+\lambda_{\min}}$$

and same holds for $|1-\eta\lambda_{\min}|$, that is

$$|1-\eta\lambda_{\min}| = \left|1 - \frac{2\lambda_{\min}}{\lambda_{\max}+\lambda_{min}}\right| = \frac{|\lambda_{\max}-\lambda_{\min}|}{\lambda_{\max}+\lambda_{\min}}$$

In fact, the value of η was chosen by equating the two expressions. ■

Although this gives us a bound on convergence of gradient descent, the theorem also tells us that large conditioning number is an inherent issue with convergence of the gradient descent method.

5.4 Fixes to The Conditioning Problem

We need to find ways to solve poorly conditioned problem that give us faster convergence. Fortunately, looking at the quadratic problem suggests a solution: we can solve the quadratic problem in a closed form.

Lemma 5.3 *If* $f(x) = \frac{1}{2}x^\intercal Ax + b^\intercal x + c$ *such that* A *is positive-definite, then* $x = -A^{-1}b$ *is the minimizer of the of* f*.*

The gradient $\nabla f = Ax + b$. Setting $\nabla f(x) = 0$ gives us $x = -A^{-1}b$. Since A is positive-definite, and $\nabla^2 f = A$, this point must be the minimizer. Note that we require positive-definite condition only to ensure that this point is a local minima.

This lemma motivates the following algorithm.

5.4.1 Newton's Method

We can adopt the method above to general functions. By Taylor's theorem we get

$$f(x + \Delta) \approx f(x) + \Delta^\intercal \nabla f(x) + \frac{1}{2}\Delta^\intercal \nabla^2 f(x)\Delta + O(\|\Delta\|^3)$$

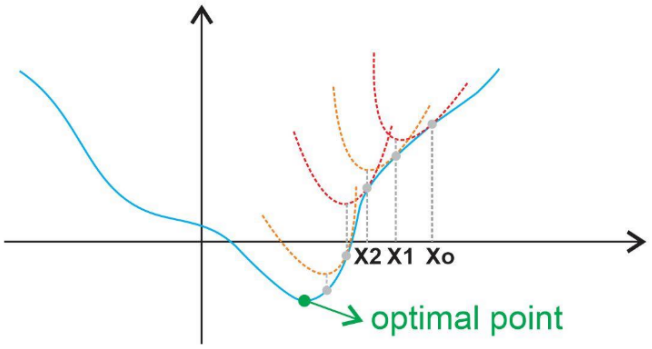
Ignoring the higher order terms, we can treat this equation as a quadratic in Δ and use the formula above to find the optimum value of Δ, which is

$$\Delta = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$$

This formula gives us the direction of the step which we need to take, giving us the update rule

$$x_{t+1} = x_t - \eta \left(\nabla^2 f(x_t)\right)^{-1} \nabla f(x_t) \tag{5.2}$$

This update rule is called as the **Newton's Method**. The figure below shows how Newton's method converges to the optimum.



Newton's method has provably better convergence than gradient descent for strongly convex function f with Lipschitz continuous second derivative. It can be shown that under these conditions, Newton's method has quadratic convergence. For details, see [BV04] pages 484-490.

Although this looks good in theory, Newton's method has a critical flaw: we need to compute Hessian of the function f, and invert it. The Hessian is $d \times d$ matrix, which we need to invert. This takes d^3 runtime, which is too expensive. Further, we will need oracle access to the Hessian, which is not always guaranteed.

This brings us back to first order methods which are computationally efficient.

5.4.2 Momentum (Polyak)

An alternative fix for bad conditioning is to use a linear combinations of the gradients at prior steps. Intuitively this smooths out the zig-zagging that can possibly happen since it doesn't rely too much on the

current gradient. This method was developed by *Boris Polyak* in 1964 in [P64]. The update rule is given by

$$v_{t+1} = -\nabla f(x_t) + \beta v_t$$
$$x_{T+1} = x_t + \eta v_{t+1}$$

This momentum has provably better guarantees than gradient descent. In particular, we have the following:

Theorem 5.4 *Let $f : \mathbb{R}^n \to \mathbb{R}$ be given by*

$$f(x) = \frac{1}{2} x^\intercal A x - b^\intercal x + c$$

where A is positive-definite matrix with condition number κ. Then there are values of β and η such that the update rules above give us sequence x_t satisfying

$$\|x_t - x^*\| \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} + \epsilon_t\right)^t \|x_0 - x^*\|$$

where $\epsilon_t \geq 0$ is a sequence such that $\lim_{t\to\infty} \epsilon_t = 0$, and x^* is the optimum solution.

5.4.3 Momentum (Nesterov)

Nesterov acceleration is a lookahead variant of momentum, first discovered by *Yutii Nesterov* in 1983 in [N83]. It updates not based on the gradient of current point, but based on the gradient of the point where gradient descent would have been had it continued in the same direction for one more step. The updates are given by

$$v_{t+1} = -\nabla f(x_t + \beta v_t) + \beta v_t$$
$$x_{t+1} = x_t + \eta v_{t+1}$$

This method is provably better for any convex function. Specifically, we have the following result:

Theorem 5.5 *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is convex and M smooth, then there exist η, β such that Nesterov's accelerated gradient descent satisfies*

$$f(x_t) - f(x^*) \leq \frac{2M\|x_0 - x^*\|^2}{t^2}$$

further, if f is m-strongly convex, then

$$f(x_t) - f(x^*) \leq \frac{m + M}{2}\left(1 - \frac{1}{\sqrt{\kappa}}\right)^{t-1} \|x_0 - x^*\|^2$$

For details, see [B14]. It turns out that this method is essentially optimal for convex function f with oracle access to the gradient. We have following result from [NY83]:

Theorem 5.6 *There is a convex and M smooth function f such that*

$$f(x_t) - f(x^*) \geq \frac{3M\|x_0 - x^*\|^2}{32(t+1)^2}$$

for any algorithm that generates x_t satisfying $x_t \in x_0 + span\{\nabla f(x_0), \cdots, \nabla f(x_{t-1})\}$ for all t.

See [N04] for proof.

5.5 BackPropagation

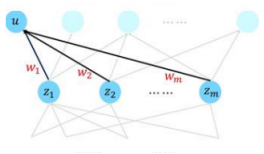
BackPropagation is the way to train neural networks. It is kind of like doing gradient descent, but on neural nets. If a neural network can be represented as a function f, then during backpropagation we want to update the weights w of each edge in the network by an amount directly proportional to the partial derivative $\frac{\partial f}{\partial w}$. Thus, we need a fast algorithm to compute the partial derivative of each edge weight to do backpropagation. The main tool here is CHAIN RULE:

Suppose $f(y) = f(x_1(y), x_2(y), \ldots, x_n(y))$

Then, $\frac{\partial f}{\partial y} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\frac{\partial x}{\partial y}$

Thus, if we want to compute the partial derivative of f w.r.t. an edge weight w_1, we can apply chain like this:

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial w_1} = \frac{\partial f}{\partial u}\frac{\partial \sigma(\langle w, z\rangle + b)}{\partial w_1}$$
$$= \frac{\partial f}{\partial u}\sigma'(u)z_1$$



Note that computing derivative of activation is easy (for most common activation functions), and we already have u and z_1, so we can compute the partial derivative of f w.r.t. w_1 in constant time if we're given the partial derivative of f w.r.t. u.

In order to compute the partial derivative of f w.r.t. the final output of a neuron u, let $y_1, \ldots y_n$ be the final output of neurons on the layer above u, since u affects all outputs of neurons on the layer above u, we get

$$\frac{\partial f}{\partial u} = \Sigma_k \frac{\partial f}{\partial y_k}\frac{\partial y_k}{\partial u} = \Sigma_k \frac{\partial f}{\partial y_k}\sigma'(y_k)w \text{ where w is the edge weight between u and } y_k$$

Thus to compute $\frac{\partial f}{\partial u}$ takes linear work w.r.t the number of neurons in the layer above u, if we're given the partial derivatives of f w.r.t. each of $y_1, y_2, \ldots$ (the neurons in layer above u). Also, note that all neurons in the same level as u will have the term $\Sigma_k \frac{\partial f}{\partial y_k}\sigma'(y_k)$ in computing the partial derivative of f with respect to them, so we only need to compute that part once. So computing partial derivatives of f w.r.t. all neurons in the same level as u will just take linear time w.r.t the sum of the number of neurons in the same layer of u and the number of neurons in the layer above u.

Thus, we've shown that if we have all partial derivatives of f with respect to all neurons in a layer, then we can compute the partial derivatives of f w.r.t. all neurons in the layer below it with runtime linear to the number of neurons in these two layers, and we can compute the partial derivatives of f w.r.t. all edge weights between the two layers each in constant time, so computing all of them takes runtime linear to the number of edges between the two layers.

So if we do a dynamic programming algorithm starting at the output level of the neural network (obviously $\frac{\partial f}{\partial f} = 1$), then we go down one layer at a time. For each level below, we compute the partial derivatives efficiently using the algorithms described above (since we already computed all partial derivatives of the neurons above them). Thus overall this algorithm computes all partial derivatives in runtime $O(|V| + |E|)$ where $|V|$ is the number of neurons in the neural network and $|E|$ is the number of edges in the neural network.

Also note that, since in this DP algorithm we really only are sharing the partial derivatives of f w.r.t the neurons, the memory required for this algorithm is going to be $O(V)$.

References

[P64] B. Polyak, "Some methods of speeding up the convergence of iteration methods", *USSR Computational Mathematics and Mathematical Physics*, 1964, URL: https://www.sciencedirect.com/science/article/abs/pii/0041555364901375

[N83] Y. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$", *Dokl. Akad. Nauk SSSR*, 1983, URL: http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=dan&paperid=46009&option_lang=eng(in russian)

[NY83] A. Nemirovskii and D. Yudin, "Problem complexity and method efficiency in optimization", *Wiley-Interscience series in discrete mathematics*, 1983

[BV04] S. Boyd and L. Vandenberghe, "Convex Optimization", *Cambridge University Press*, 2004, URL: https://dl.acm.org/doi/book/10.5555/993483

[N04] Y. Nesterov, "Introductory Lectures on Convex Optimization", *Springer(US)*, 2004, URL: https://link.springer.com/book/10.1007%2F978-1-4419-8853-9

[B14] S. Bubeck, "Convex Optimization: Algorithms and Complexity", 2014, URL: https://arxiv.org/abs/1405.4980