Lecture 10: Gradient-based optimization

Tushaar Gangavarapu (TG352@cornell.edu)

So far in this course, we have seen several algorithms for supervised and unsupervised learning. For most of these algorithms, we wrote down an optimization objective—either as a cost function (in k-means, mixture of Gaussians, principal component analysis) or log-likelihood function, parameterized by some parameters. In essence, we hoped to find the *optimal* parameters by minimizing (or maximizing) the objective. For notational convenience, we will denote the optimization objective as $J(\theta)$, parametrized by θ.

For most of these algorithms, we found closed-form solutions—set the derivative of $J(\theta)$ to zero and solve for θ—to our objective. In the previous lecture, we saw logistic regression, where the objective (to minimize) was:

$$J(\theta) = \sum_{j=1}^{n} \log(1 + \exp(-y^{(j)}\theta^T x^{(j)})),$$

where $x^{(j)} \in \mathbb{R}^d$ is the j-th training sample with $y^{(j)} \in \{+1, -1\}$ as the associated label, and n is the total number of training samples. Now, if you try to solve for θ by taking the derivative with respect to θ and set it to zero, you will quickly realize that it is not possible to estimate θ in closed form from above; see Appendix B for proof. Even in cases where an analytical solution is derivable, it can be computationally expensive to compute the exact solution.[1]

In this lecture, we will focus on *iterative* approaches to finding the optimal θ. The ideas discussed here will apply broadly and are not limited to a specific algorithm. Our goal is to choose θ to minimize $J(\theta)$, and we want to do it iteratively.[2] Let us formalize this: we wish to construct a sequence of iterates, $\theta^{(1)}, \ldots, \theta^{(k)}$, starting with a *good* initial guess $\theta^{(0)} \in \mathbb{R}^d$ such that,

$$\theta^{(k+1)} = G(\theta^{(k)}),$$

where G is our iteration. Observe that, upon convergence to optimal $\theta^\star$, we require $G(\theta^\star) = \theta^\star$; in other words, $\theta^\star$ is the *fixed point* of iteration G. This rather simple formulation is quite powerful in convergence analysis of G, as we will see later.

Significance. Gradient descent and its variants have been fundamental in powering modern large language models with complex, non-convex landscapes. Beyond computational efficiency, gradient descent has been shown to have implicit regularization effects—see Landweber iteration[3] and double descent phenomenon (Belkin et al., 2019) for specifics. We will uncover the core ideas in double descent phenomenon when we discuss bias-variance tradeoff.

1 Gradient descent

So we wish to form our sequence of iterates, $\theta^{(1)}, \ldots, \theta^{(k)}$, starting from a good initial guess $\theta^{(0)}$.[4] As an example, let us consider a simple $J(\theta)$ as follows:

$$J(\theta) = \theta_1^2 + \theta_2^2,$$

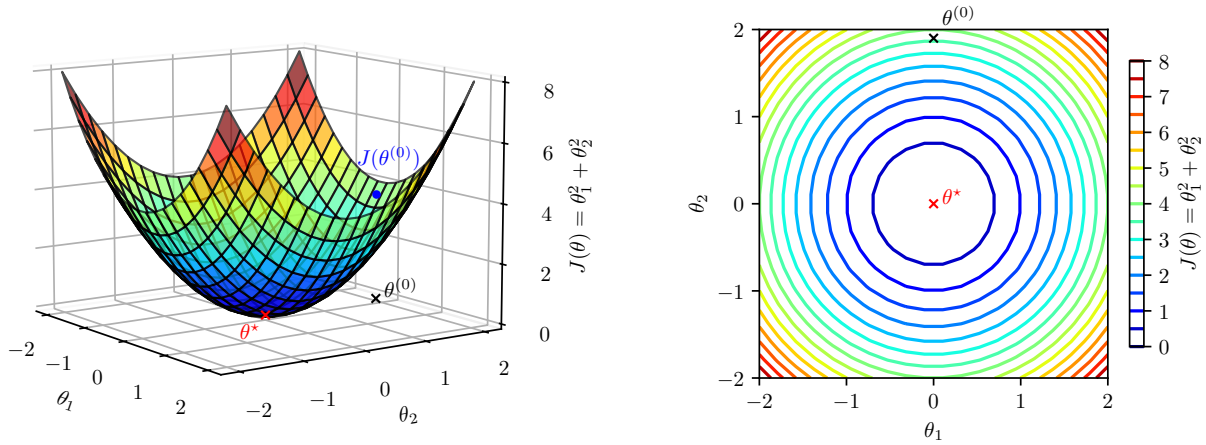
[1] We will see an example of this when we discuss linear regression.

[2] We will only deal with scalar-valued functions, i.e., $J : \mathbb{R}^d \to \mathbb{R}$; vector-valued functions, i.e., $J : \mathbb{R}^d \to \mathbb{R}^m$, are out of scope for this lecture. See §4.2 of Gangavarapu (2023) for further reading.

[3] https://www.cs.cornell.edu/courses/cs6241/2023fa/lec/2023-08-31.pdf.

[4] We will discuss the impact of choice of $\theta^{(0)}$ and convergence of G later, and if it even matters to finding the optimal $\theta^\star$ later; for now, we will assume a reasonably good initial guess.

where $\theta \in \mathbb{R}^2$; we use subscripts to indicate dimension indices. The landscape of $J(\theta)$ is as follows; 3D plot to the left and contour plot, a.k.a., top view, to the right:



Observe that $\theta^\star = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ is the minimizer of $J(\theta)$, and $J(\theta^\star) = 0$.

Now, the goal is to come up with an iteration, G, such that we can iteratively converge to $\theta^\star$, starting from $\theta^{(0)}$. Recall that we wish to construct G such that $\theta^\star$ is a fixed point G.

1.1 Gradient and ascent direction

Let us briefly recall what (partial) derivatives tell us: they indicate the behavior of a function in an infinitesimally-small region around a given point. Simply put, for some function of x, $f(x)$, we have

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=x^{(0)}} = f'(x^{(0)}) = \frac{f(x^{(0)}+h) - f(x^{(0)})}{h},$$

for some small h in the order of $\mathcal{O}(10^{-5})$, which tells us the behavior of f around $x^{(0)}$. This is often referred to as the *first* order approximation of the derivative, and incurs an error in the order of $\mathcal{O}(h)$.[5]

Given this notion, we are now interested in interpreting what a derivative tells us. Consider our previous example of $J(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$ and we'll look at the behavior of this function at $(\theta_1^{(0)}, \theta_2^{(0)}) = (0, 2)$:

$$\left.\frac{\partial J}{\partial \theta_1}\right|_{(\theta_1,\theta_2)=(0,2)} = 2\theta_1^{(0)} = 0; \qquad \left.\frac{\partial f}{\partial \theta_2}\right|_{(\theta_1,\theta_2)=(0,2)} = 2\theta_2^{(0)} = 4;$$

Recall from our definition of derivative, we have:

$$f(\theta_1^{(0)} + h) = f(\theta_1^{(0)}) + \frac{\partial f}{\partial \theta_1} h = f(\theta_1^{(0)}),$$

which indicates that *small* perturbations to θ_1 (at $\theta_1 = 0$) doesn't change the value of J. Similarly, we have:

$$f(\theta_2^{(0)} + h) = f(\theta_2^{(0)}) + \frac{\partial f}{\partial \theta_2} h = f(\theta_2^{(0)}) + 4h,$$

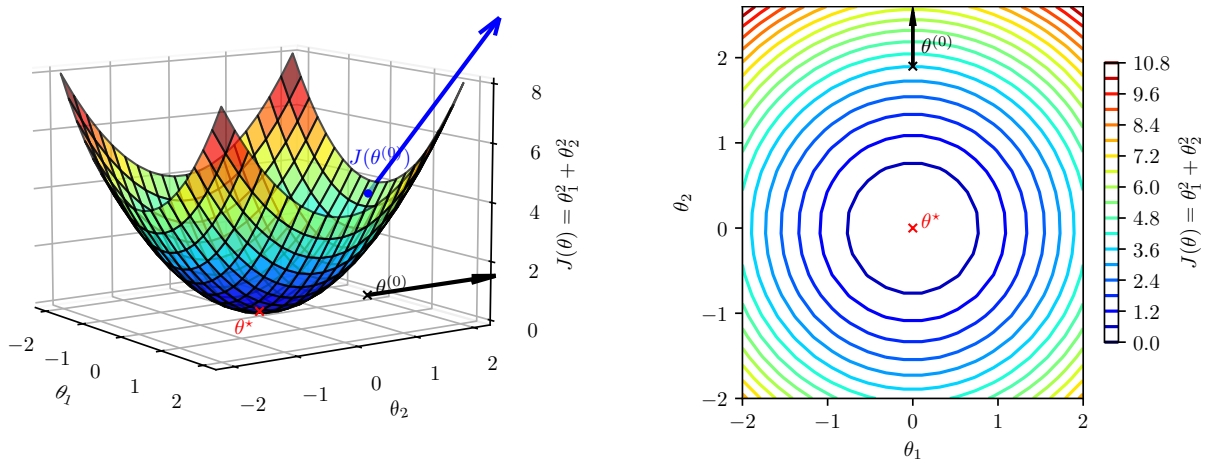
[5] For the purposes of this lecture, we only use this difference-based definition of a derivative (also referred to as *numerical* differentiation) for intuition. In practice, numerical differentiation is ofter performed using a symmetric and centered difference, which is a second-order approximation. See Appendix C for details.

meaning: (small) perturbations to θ_2 (at $\theta_2 = 2$) are amplified $4\times$ in J. Hence, derivative is an indicator of how sensitive the function, J, is to the changes in the input, θ_1, θ_2.

The key observation here is that this sensitivity measure acts a dial which we can use to navigate the J landscape. In the above example, starting from $(\theta_1, \theta_2) = (0, 2)$ and moving along the direction:

$$\begin{bmatrix} \dfrac{\partial J}{\partial \theta_1} \\ \dfrac{\partial J}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix},$$

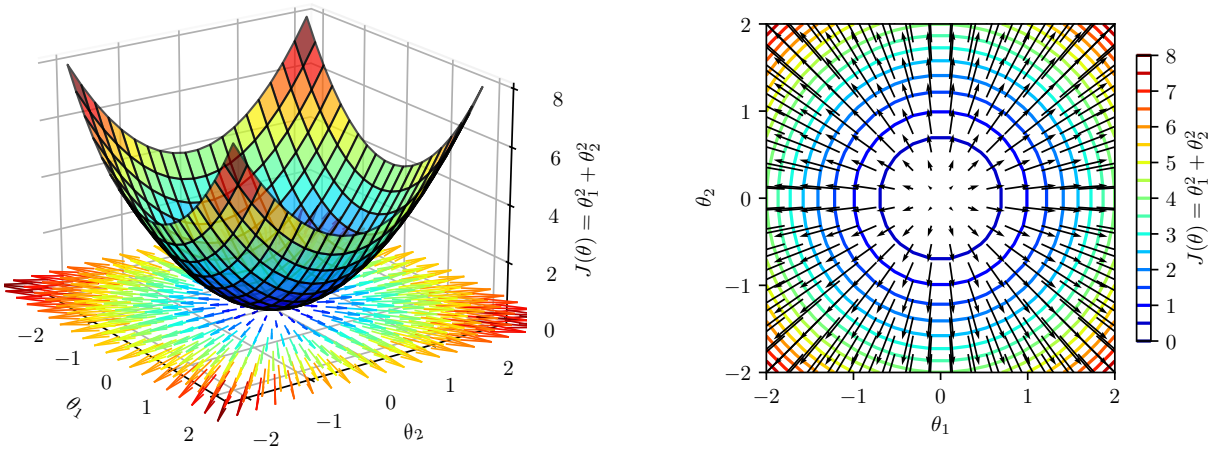
results in the greatest increase in J:



For convenience, we will collect partial derivatives of J with respect all θ_is into a vector, $\nabla J(\theta^{(k)})$, read "nabla J" or "del J":

$$\nabla J(\theta^{(k)}) = \begin{bmatrix} \left.\dfrac{\partial J}{\partial \theta_1}\right|_{\theta=\theta^{(k)}} \\ \vdots \\ \left.\dfrac{\partial J}{\partial \theta_d}\right|_{\theta=\theta^{(k)}} \end{bmatrix},$$

which we call the *gradient* vector. Notice that gradient indicates the *steepest* uphill direction:



One final thing to note: observe J at $\theta^\star$ (here, $(\theta_1, \theta_2) = (0, 0)$); it is "flat" in all directions, i.e., there is no one steepest ascent direction. Mathematically, this means that the partial derivatives, which form ∇J, are all zero at $\theta^\star$. Hence, we have $\nabla J(\theta^\star) = 0$.

1.2 Gradient descent iteration

Given that gradient indicates the steepest ascent direction, it follows that moving in the opposite direction should give us the steepest descent direction, eventually bringing us to $\theta^\star$. We now write down the gradient descent or steepest descent iteration as:

$$\theta^{(k+1)} = G(\theta^{(k)}) = \theta^{(k)} + \alpha \underbrace{(-\nabla J(\theta^{(k)}))}_{\substack{\text{steepest descent} \\ \text{direction}}},$$

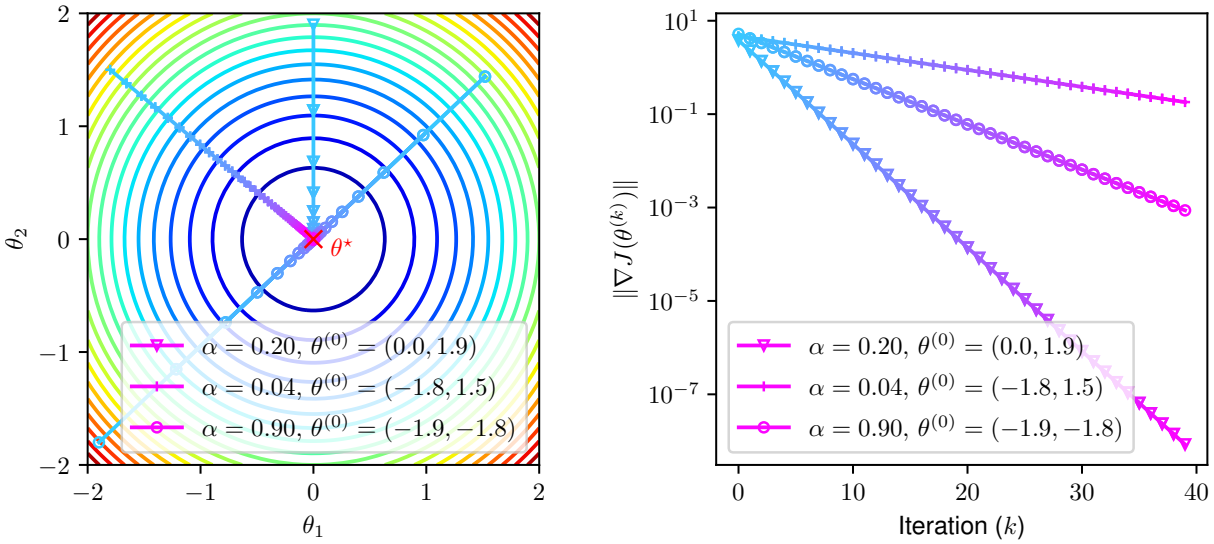
where learning rate, α, is a hyperparameter that controls how big a step to take in the steepest descent direction. Recall that for $\theta^{(k)} = \theta^\star$, we have

$$\theta^{(\star+1)} = G(\theta^\star) = \theta^\star + \alpha(-\underbrace{\nabla J(\theta^\star)}_{=0}) = \theta^\star.$$

Hence, we observe convergence upon reaching $\theta^{(k)} = \theta^\star$, the optimal values of the model parameters. It is often the case that we run gradient descent for a fixed number of iterations or until a suffiently small gradient is observed (since gradient at $\theta^\star$ is zero).

It is important to explicitly note here that we made an implicit assumption of a "nice" convex J, i.e., specifically that J is differentiable at least once and the first derivative is continuous.[6]

We can now run the gradient descent iteration on our previous example with different configurations of α and starting point, $\theta^{(0)}$:



It is easy to see from above that α controls the rate of convergence. It is also quite clear that a relatively small α implies slow convergence (e.g., $\alpha = 0.04$ above). Be warned: it is inaccurate to conclude that a large α gives us faster convergence; see $\alpha = 0.9$ setting above, where the values of θ bounce between two walls of the bowl. Now, a natural question is to ask if there are settings for α, where we *do not* converge.[7]

[6] This is often denoted as $\mathscr{C}^1$-continuous, or more generally, $\mathscr{C}^k$-continuous, to indicate that that the function is differentiable k times and the k-th derivative is continuous.

[7] Interactive demo for $J(\theta) = \theta_1^2 + \theta_2^2$: https://colab.research.google.com/drive/14G5hZpzHGOGr3wi_fE7hJanYsvSiiqFi.

1.3 On the convergence of gradient descent

While the proof of convergence of gradient descent is out of the scope of this class, we wish to show, or at the very least, analyze the convergence for $J(\theta) = \theta_1^2 + \theta_2^2 = \theta^T\theta$. Let's take this a step further and show convergence for a general quadratic of the form:

$$J(\theta) = \frac{1}{2}\theta^T A\theta + b^T\theta + c,$$

for some symmetric positive definite $A \in \mathbb{R}^{d\times d}$, $b \in \mathbb{R}^d$, and $c \in \mathbb{R}$.[8] We have ∇J as

$$\nabla J = \frac{1}{2}(A + A^T)\theta + b = A\theta + b;$$

here, we used the fact that $A = A^T$ for a symmetric A.[9] Now, we have:

$$\begin{aligned}
\theta^{(k+1)} &= \theta^{(k)} - \alpha(A\theta^{(k)} + b) \\
- \quad \theta^{\star+1} &= \theta^\star = \theta^\star - \alpha(A\theta^\star + b) \\
\hline
\theta^{(k+1)} - \theta^\star &= (\theta^{(k)} - \theta^\star) - \alpha(A(\theta^{(k)} - \theta^\star)) \\
&= (\mathrm{I} - \alpha A)(\theta^{(k)} - \theta^\star).
\end{aligned}$$

Now, what the above tells us is that error at step $k+1$, $\varepsilon^{(k+1)} = \theta^{(k)} - \theta^\star$ grows or decays by a factor of $\mathrm{I} - \alpha A$, compared to the error at step k, $\varepsilon^{(k)} = \theta^{(k)} - \theta^\star$. We can expect convergence if,

$$\|\varepsilon^{(k+1)}\| < \|\varepsilon^{(k)}\|, \text{ OR equivalently, } \|(\mathrm{I} - \alpha A)\varepsilon^{(k)}\| < \|\varepsilon^{(k)}\|.$$

Clearly, we can see that the maximum "stretch" of the transformation $\mathrm{I} - \alpha A$ determines the rate at which $\varepsilon^{(k)}$ contracts. Recall that the maximum stretch is governed by the (magnitude of the) largest eigenvalue of $\mathrm{I} - \alpha A$ (also known as the spectral radius). Hence, for convergence, we require

$$\max_j |1 - \alpha\lambda_j| < 1,$$

where λ_j is an eigenvalue of A. Expanding this, we have:

$$-1 < 1 - \alpha\lambda_j < 1, \text{ or, } 0 < \alpha < 2/\lambda_j,$$

for all λ_js. If we arrange all eigenvalues by magnitude from $\lambda_{\min}, \ldots, \lambda_{\max}$, we get bounds for α as $2/\lambda_{\min}, \ldots, 2/\lambda_{\max}$. Now, observe that is $2/\lambda_{\max}$ is the largest possible α value that satisfies $\alpha < 2/\lambda_j$ for all j. Hence, for convergence we require $\alpha < 2/\lambda_{\max}$.

In our example of $J(\theta) = \theta_1^2 + \theta_2^2 = \theta^T(A/2)\theta$, where $A = 2\mathrm{I}$ (all eigenvalues are 2), we require $\alpha < 1$ for convergence. You can verify this to be true by running the demo script above[7] with $\alpha > 1$.

Aside on the quadratic form. Recall that A is a positive definite matrix iff $x^T Ax = \langle x, Ax\rangle > 0$ for all $x \neq 0$, i.e., both x and Ax (a transformation on x) point in the same direction for all x.

From Taylor's theorem around $\theta^\star$, we have:

$$J(\theta^\star + h) = J(\theta^\star) + \nabla J(\theta^\star)^T h + \frac{1}{2}h^T\nabla^2 J(\theta^\star)h + \mathcal{O}(h^3).$$

[8] For an asymmetric A, we can write $A = (A+A^T)/2 + (A-A^T)/2$. It is easy to show that $\langle x, (A-A^T)x\rangle = 0$ for all x. Thus, for every asymmetric A, we have $B = (A + A^T)/2$, such that $\langle x, Ax\rangle = \langle x, Bx\rangle$. Hence, we assume a symmetric matrix A; if not, we can convert it to an equivalent quadratic form with a symmetric matrix.

[9] For those less familiar with matrix derivatives, The Matrix Cookbook serves as a good reference: https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf.

Now, since $\theta^\star$ is a point of local optima, the gradient is zero (recall: no ascent direction; the function is flat everywhere), i.e, $\nabla J(\theta^\star) = 0$. Hence we have:

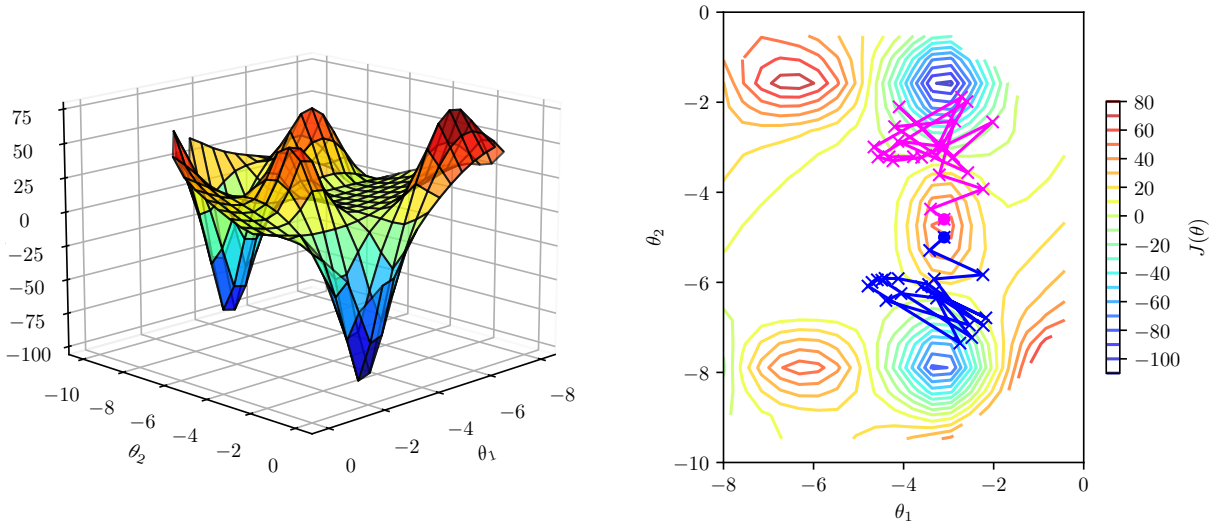
$$J(\theta^\star + h) - J(\theta^\star) \approx \frac{1}{2} h^T \nabla^2 J(\theta^\star) h.$$

Realize from the general quadratic form above that $\nabla^2 J(\theta^\star) = A$ for a symmetric A. Hence we have $J(\theta^\star + h) - J(\theta^\star) > 0$ or $J(\theta^\star + h) > J(\theta^\star)$ for any h, which implies that J is strictly convex.

1.4 "Good" initial guess

One question remains unanswered: what is a reasonably good initial guess $\theta^{(0)}$?

The above convergence analysis marks that for a sufficiently small α, gradient descent converges. This guarantee holds for any initial guess, so long as there is a unique global minimum. As a motivating example, consider the following landscape for $J(\theta)$:



Observe that two nearby starting points (marked in magenta and blue respectively on the figure to the right) lead to different minima. Note that both these minima are fixed points of our gradient descent iteration, i.e., $\nabla J = 0$ at both these minima.

A common strategy that is often used with rather simple cost functions is to run gradient descent multiple times from different starting points and choosing the the minimizer from the configurations. This can get really expensive in modern machine learning models where we need to estimate billions of parameters (for an estimate: Llama-3.1 has 405B parameters). In the next lecture, we will explore a slightly different approach, where we (hopefully) jolt out of local minima by leveraging a noisy gradient estimate.

2 Newton's method

In the previous section, we saw how moving along the steepest descent direction to eventually reach the fixed point, $\theta^\star$, where $\nabla J(\theta^\star) = 0$. Hence, finding an iteration that iteratively converges to $\theta^\star$ can be reformulated as finding the solution, $\theta^\star$ to $\nabla J = 0$ *iteratively*. Now, this reformulation should seem natural to you—after all, this is exactly how we find the optimal value; take the derivative, set it to zero, and solve for $\theta^\star$. All that we seek is to come up with an iterative approach to doing this.

2.1 Newton's method for root finding

We will come back to the exact problem later, but the essence of it is to find the roots of a function, $f(\theta)$ iteratively. Let's assume a simple 1D function for the moment; we will generalize this later. Using Taylor's theorem around $\theta^{(k)}$, we can write

$$f(\theta) = f(\theta^{(k)}) + f'(\theta^{(k)})(\theta - \theta^{(k)}) + \mathcal{O}((\theta - \theta^{(k)})^2).$$

Now, let's use the above to evaluate f at $\theta^{(k+1)}$:

$$f(\theta^{(k+1)}) = f(\theta^{(k)}) + f'(\theta^{(k)})(\theta^{(k+1)} - \theta^{(k)}) + \mathcal{O}((\theta^{(k+1)} - \theta^{(k)})^2).$$

Observe that if $\theta^{(k+1)}$ and $\theta^{(k)}$ are close enough, we can ignore the squared term above: it will be relatively small. Next, *if* $\theta^{(k+1)}$ is a root of f, we have $f(\theta^{(k+1)}) = 0$. Hence, the above reduces as follows:

$$f(\theta^{(k+1)}) \approx f(\theta^{(k)}) + f'(\theta^{(k)})(\theta^{(k+1)} - \theta^{(k)}) \stackrel{\text{set}}{=} 0. \tag{1}$$

Rearranging the above gives us:

$$\theta^{(k+1)} = \theta^{(k)} - \frac{f(\theta^{(k)})}{f'(\theta^{(k)})}.$$

One might ask: does the above mean that we are going to find the root in a single step? No, realize that the above formulation is modeled based on the "local" behavior near $\theta^{(k)}$, modeling using our first-order approximation—we discarded the $\mathcal{O}((\theta^{(k+1)} - \theta^{(k)})^2)$ term. Okay, a natural follow up is to ask how far away is $\theta^{(k+1)}$ from $\theta^\star$, if $\theta^\star$ is the root of f, i.e., $f(\theta^\star) = 0$. Again, from Taylor's theorem, we have

$$0 = f(\theta^\star) = f(\theta^{(k)}) + f'(\theta^{(k)})(\theta^\star - \theta^{(k)}) + \mathcal{O}((\theta^\star - \theta^{(k)})^2). \tag{2}$$

If we subtract the Taylor expansion of $f(\theta^{(k+1)})$ from $f(\theta^\star)$, i.e., (2) − (1), we get:

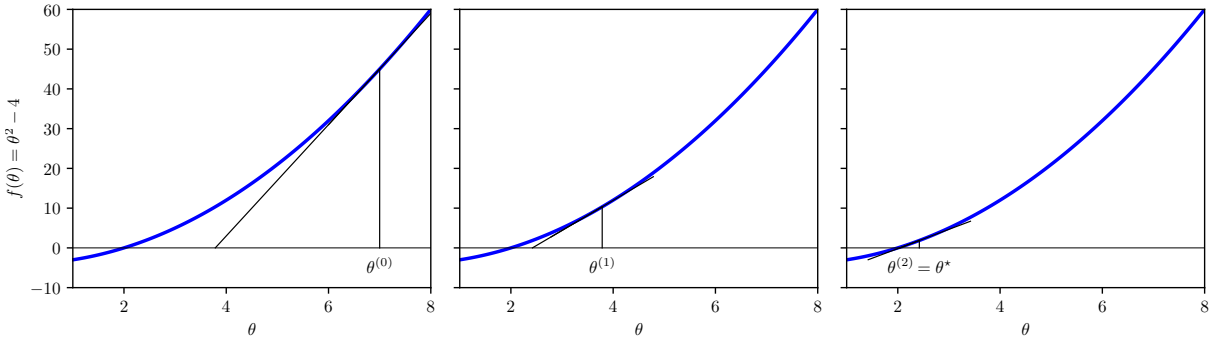
$$0 = f'(\theta^{(k)}) \underbrace{(\theta^\star - \theta^{(k+1)})}_{=\varepsilon^{(k+1)}} + \mathcal{O}((\underbrace{\theta^\star - \theta^{(k)}}_{=\varepsilon^{(k)}})^2).$$

Now, the above tells us that error at step $k+1$, $\varepsilon^{(k+1)} = \theta^\star - \theta^{(k+1)}$ decays quadratically:

$$\varepsilon^{(k+1)} = -\frac{\mathcal{O}({\varepsilon^{(k)}}^2)}{f'(\theta^{(k)})} = C{\varepsilon^{(k)}}^2,$$

assuming some modest constant, C. This implies that, a relative small error at step k leads to a *really* small error at step $k+1$. Such behavior, where the errors is squared at each step is known as *quadratic convergence*.

Hence, Newton's method gives us a series of iterates, $\theta^{(1)}, \ldots, \theta^{(k)}$, starting from $\theta^{(0)}$, with quadratic convergence properties. Let us see what this looks like:



We interpret the above as follows: fit a tangent (first-order approximation) to the given function at the current guess for θ, find its zero, and use it as the next guess for θ. We draw attention to the fact that the Newton's method has converged in just three steps!

2.2 From root finding to optimization

Recall that our optimization goal is to iteratively find roots of ∇J, or f' in 1D. We can use the Newton root-finding approach discussed above, with the function being $f'(\theta)$. Hence, we have:

$$\theta^{(k+1)} = \theta^{(k)} - \frac{f'(\theta^{(k)})}{f''(\theta^{(k)})},$$

which is the Newton's iteration for optimization. The above iteration gives us a sequence of iterates that eventually find $\theta^\star$ such that $f'(\theta^\star) = 0$. Realize that the convergence in best case scenario (a modest constant, C) for Newton's iteration is quadratic, as opposed to linear for gradient descent at best.

Lastly, in our case of a scalar-valued J and vector-valued θ, we can generalize Newton's iteration as:

$$\theta^{(k+1)} = \theta^{(k)} - H_J(\theta^{(k)})^{-1}\nabla J(\theta^{(k)}),$$

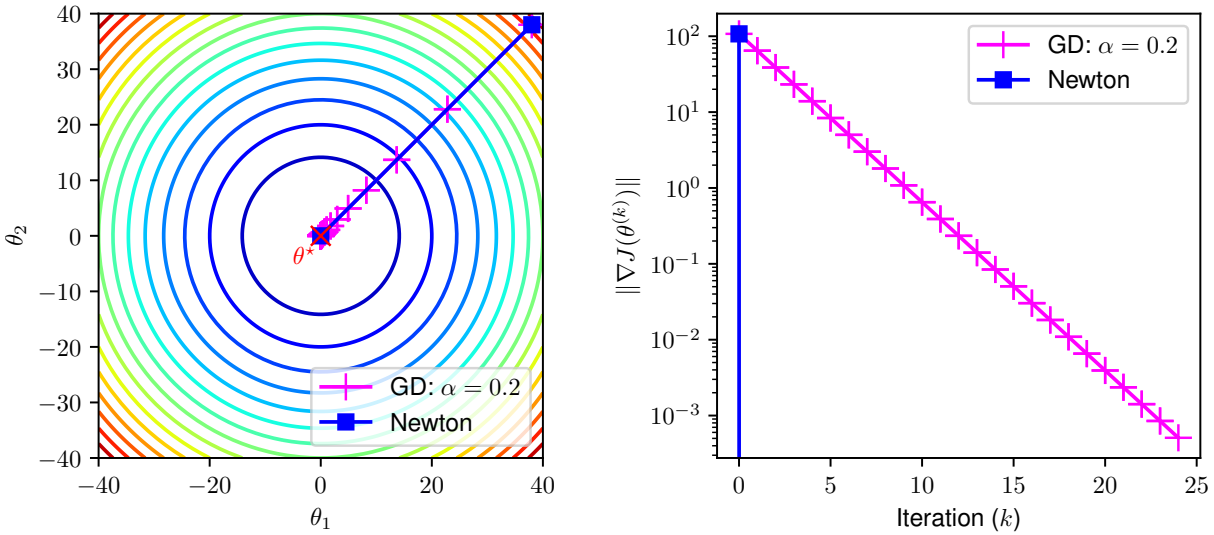
where $\nabla J(\theta^{(k)})$ is our vector of partial derivatives, evaluated at $\theta^{(k)}$ and $H_J(\theta^{(k)})$ is an $d \times d$ matrix that collects the second-order derivatives. For $\theta \in \mathbb{R}^2$, we have

$$H_J(\theta^{(k)}) = \begin{bmatrix} \frac{\partial^2 J}{\partial \theta_1^2}\Big|_{\theta=\theta^{(k)}} & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2}\Big|_{\theta=\theta^{(k)}} \\ \frac{\partial^2 J}{\partial \theta_2 \partial \theta_1}\Big|_{\theta=\theta^{(k)}} & \frac{\partial^2 J}{\partial \theta_2^2}\Big|_{\theta=\theta^{(k)}} \end{bmatrix}.$$

In general, H_J is called the "Hessian" of function J and the (i,j)-th entry of H_J is given by:

$$(H_J)_{i,j} = \frac{\partial}{\partial \theta_i}\left[\frac{\partial J}{\partial \theta_j}\right] = \frac{\partial^2 J}{\partial \theta_i \partial \theta_j}.$$

We can now run Newton's iteration on our previous example of $J(\theta) = \theta_1^2 + \theta_2^2$ and contrast it with gradient descent:



Depending on how you look at it, it's either a "huh?" or an "aha!" moment—the Newton iteration converged in a single iteration! We can choose any starting point and Newton is

guaranteed to converge in a single iteration for $J(\theta) = \theta_1^2 + \theta_2^2$. For this J, we have

$$\nabla J(\theta^{(0)}) = \begin{bmatrix} 2\theta_1^{(0)} \\ 2\theta_2^{(0)} \end{bmatrix} = 2\theta^{(0)}; \qquad H_J(\theta^{(0)})^{-1} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix},$$

which gives

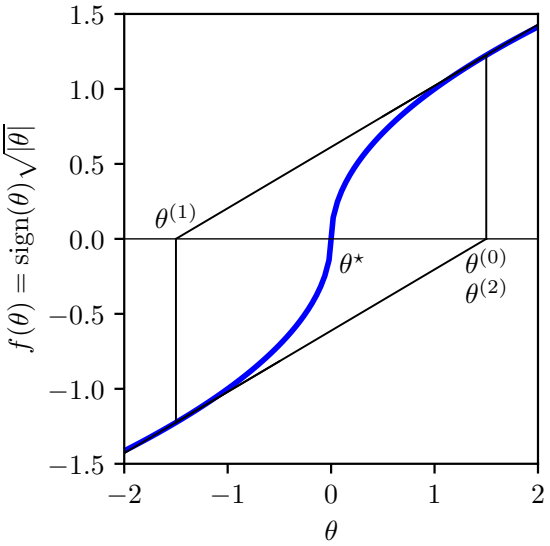
$$\theta^{(1)} = \theta^{(0)} - H_J^{-1}\nabla J = \theta^{(0)} - \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} (2\theta^{(0)}) = \theta^{(0)} - \theta^{(0)} = 0.$$

Hence, for our J, the Newton's method jumps directly to the minimum in one iteration.

One final thing to note: similar to gradient descent, we implicitly assumed a "nice" convex J, i.e., J is differentiable at least twice and the second derivative is continuous.

2.3 Always run Newton?

There are several reasons for not wanting to run Newton iteration to find the optimum. An obvious one is when Hessian is not well-behaved, i.e., what if we can't invert the Hessian? Or, if Hessian is near-singular, the inversion becomes unstable. Additionally, at saddle points, Newton's iteration may move away from the optimum than towards it. Sometimes, the iteration can simply oscillate between two points; an example of this behavior in root finding is shown to the right.



Finally, one iteration of Newton is more expensive than one iteration of gradient descent, since it requires computing and inverting an $d \times d$ Hessian. So long as d is not significantly large, Newton usually enjoys faster convergence.

3 Conclusion

In this lecture, we motivated the need for *iterative* methods to finding parameters that minimize a given cost function. We then saw how gradient models the steepest ascent direction, and realized that taking "sufficiently small" steps in the steepest descent direction eventually leads us to a minimum (not necessarily the global minimum). Furthermore, we analyzed the convergence properties of gradient descent for a general quadratic and noted linear convergence for "nice" J.

Next, we realized how our minimization problem could be reformulated as one of root finding, following which, we saw the Newton's method for optimization. We then contrasted Newton with gradient descent to note the quadratic convergence rate of Newton for nice functions. Finally, we looked at why one might not want to run Newton optimization. Of course, it is also possible to devise an iteration in between gradient descent and Newton, and such approaches are often termed as "scaled" gradient descent iterations.

In the next lecture, we will explore the limitations of gradient descent and develop strategies to overcome them.

A Notation

d	The number of parameters we wish to estimate. Note that we also use d to represent the number of features; these need not be the same, as we will see later in the course
$x^{(j)} \in \mathbb{R}^d$	The d-dimensional feature vector associated with the j-th training sample
$y^{(j)}$	The class label associated with the j-th training sample
$x_\ell^{(j)} \in \mathbb{R}$	The ℓ-th element of $x^{(j)}$
$\theta \in \mathbb{R}^d$	A vector of d model parameters
$\theta_\ell \in \mathbb{R}$	The ℓ-th element of θ
$J(\theta)$	The cost function we are trying to optimize (here, minimize)
$\theta^{(k)} \in \mathbb{R}^d$	The k-th iterate of model parameters
$\theta^\star \in \mathbb{R}^d$	The optimal model parameters that minimize $J(\theta)$
G	The iteration that updates the parameters based on their current values. At the optima, $\theta^\star$, we require $G(\theta^\star) = \theta^\star$
$f'(\theta^{(k)}) = \mathrm{d}f(\theta^{(k)})/\,\mathrm{d}\theta$	The derivative of $f(\theta)$ with respect to θ, evaluated at $\theta^{(k)}$. Upright d is different from the dimension d
$\partial f(\theta^{(k)})/\partial\theta_i$	The *partial* derivative of $f(\theta_i, \dots)$ with respect to θ_i, evaluated at $\theta^{(k)}$
$h \sim \mathcal{O}(10^{-5})$	An infinitesimally-small perturbation around a given point, often used in the definition of the derivative
$\nabla J(\theta^{(k)}) \in \mathbb{R}^d$	A vector of partial derivatives of $J(\theta)$, evaluated at $\theta^{(k)}$, known as the gradient vector (read: "nabla J" or "del J"). Advanced: If you're familiar with Jacobian, gradient is the dual of Jacobian for scalar-valued functions.
α	The learning rate or step size used to move along the steepest descent direction in gradient descent
$\varepsilon^{(k)}$	Error at step k, computed as $\theta^\star - \theta^{(k)}$
$\|u\|$	The (two) norm of a vector u
λ_j	The j-th eigenvalue of a matrix
$\lambda_{\min}$	The smallest eigenvalue of a matrix
$\lambda_{\max}$	The largest eigenvalue of a matrix
Taylor exp. $f(\theta)$	For a 1D function, f, $f(\theta) = f(\theta^{(k)}) + f'(\theta^{(k)})(\theta - \theta^{(k)}) + \mathcal{O}((\theta - \theta^{(k)})^2)$
$f''(\theta^{(k)}) = \mathrm{d}^2 f(\theta^{(k)})/\,\mathrm{d}\theta^2$	The second derivative of $f(\theta)$ with respect to θ, evaluated at $\theta^{(k)}$.
$\partial^2 f(\theta^{(k)})/\partial\theta_i\partial\theta_j$	The (second order) partial derivative of $f(\theta_i, \theta_j, \dots)$ with respect to θ_i and θ_j, evaluated at $\theta^{(k)}$
$H_J(\theta^{(k)}) = \nabla^2 J(\theta^{(k)}) \in \mathbb{R}^{d\times d}$	The matrix of second order partial derivatives of $J(\theta)$, evaluated at $\theta = \theta^{(k)}$, with the (i,j)-th entry being the partial with respect to θ_i and θ_j. This is often called the Hessian

B Optimizing logistic cost function in closed form

Given the logistic cost function,

$$J(\theta) = \sum_{j=1}^{n} \log(1 + \exp(-y^{(j)}\theta^T x^{(j)})),$$

we want to show that obtaining a closed-form solution for θ is unfeasible. We follow the standard procedure of taking the derivative with respect to θ and setting it to zero, to solve for

optimal θ:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{j=1}^{n} \frac{\partial}{\partial \theta} \log(1 + \exp(-y^{(j)}\theta^T x^{(j)})) \\ &= \sum_{j=1}^{n} \frac{-y^{(j)}x^{(j)} \exp(-y^{(j)}\theta^T x^{(j)})}{1 + \exp(-y^{(j)}\theta^T x^{(j)})} \\ &= \sum_{j=1}^{n} \frac{-y^{(j)}x^{(j)}}{\exp(y^{(j)}\theta^T x^{(j)}) + 1} \stackrel{\text{set}}{=} 0.\end{aligned}$$

One can clearly see that the complexity introduced by the nonlinear exponential terms above prevents us from computing the optimal θ in closed form. This motivates that not all convex functions[10] have closed-form solutions.

C Numerical differentiation and Taylor approximations

We defined the derivative of a function f at a point $x^{(0)}$, $f'(x^{(0)})$ using the finite difference as:

$$f'(x^{(0)}) = \frac{f(x^{(0)} + h) - f(x^{(0)})}{x^{(0)}},$$

for some relatively small perturbation, h, in the order of $\mathcal{O}(10^{-5})$. We will now proceed to show how this is resultant of a first order approximation of $f(x)$ around $x^{(0)}$.

From Taylor's remainder therem, $f(x)$ around $x^{(0)}$ can be realized as

$$f(x) = f(x^{(0)}) + f'(x^{(0)})(x - x^{(0)}) + \underbrace{\frac{1}{2}f''(\xi)(x - x^{(0)})^2}_{\text{remainder}},$$

for some ξ in between $x^{(0)}$ and x. Using remainder is a more rigorous than simply writing $\mathcal{O}((x - x^{(0)})^2)$, but they drive the same point. Now, if we evaluate $f(x)$ at $x = x^{(0)} + h$ for some small perturbation h, we get

$$f(x^{(0)} + h) = f(x^{(0)}) + f'(x^{(0)})h + \frac{1}{2}f''(\xi)h^2,$$

which gives us

$$f'(x^{(0)}) = \frac{f(x^{(0)} + h) - f(x^{(0)})}{h} - \frac{1}{2}f''(\xi)h.$$

If we ignore the "$(1/2)f''(\xi)h$" term above, i.e., essentially resorting to a first-order approximation, we incur an $\mathcal{O}(h)$ error in our estimation of the derivative. We can further refine this approximation by considering a centered and symmetric difference—for some ξ between $x^{(0)}$ and $x^{(0)} + h$ and ζ between $x^{(0)} - h$ and $x^{(0)}$, we have:

$$\begin{aligned}f(x^{(0)} + h) &= f(x^{(0)}) + f'(x^{(0)})h + \frac{1}{2}f''(x^{(0)})h^2 + \frac{1}{6}f'''(\xi)h^3; \\ f(x^{(0)} - h) &= f(x^{(0)}) - f'(x^{(0)})h + \frac{1}{2}f''(x^{(0)})h^2 - \frac{1}{6}f'''(\zeta)h^3.\end{aligned}$$

[10] We leave it as a self-exercise to show that the logistic cost function is convex.

Now, subtracting the above equations leaves us with

$$f(x^{(0)} + h) - f(x^{(0)} - h) = 2f'(x^{(0)})h + \frac{1}{6}(f'''(\xi) + f'''(\zeta))h^3,$$

which gives us the derivative as

$$f'(x^{(0)}) = \frac{f(x^{(0)} + h) - f(x^{(0)} - h)}{2h} - \frac{1}{12}(f'''(\xi) + f'''(\zeta))h^2.$$

Same as before, if we use the second-order approximation and ignore higher-order terms resulting in an $\mathcal{O}(h^2)$ error, we have

$$f'(x^{(0)}) = \frac{f(x^{(0)} + h) - f(x^{(0)} - h)}{2h},$$

which is often known as the "symmetric difference" version.

References

M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, jul 2019. doi: 10.1073/pnas.1903070116. URL https://doi.org/10.1073%2Fpnas.1903070116.

D. Bindel. CS4220 Lecture notes on Gradient descent and Newton for Optimization. *CS4220 Lecture notes*, 1(1):1–12, 2023. URL https://www.cs.cornell.edu/courses/cs4220/2023sp/lec/2023-03-31.pdf.

T. Gangavarapu. CS 4740 Fa'23: Lecture notes on gradient-based optimization and automatic differentiation, October 2023. URL https://github.com/TushaarGVS/backprop-lecture-notes-CS-4740. published 10/03/2023, revised 10/08/2023.

G. Peyré. Course notes on optimization for machine learning. *Notes de cours de lÉcole Normale Supérieure*, pages 4–10, 2020. URL https://mathematical-tours.github.io/book-sources/optim-ml/OptimML.pdf.

(Last compiled: 3/4/2025, 11.03am ET.)