3 Gradient Descent

> **Note**
>
> This page contains all content from the legacy [PDF notes](); gradient descent chapter.
>
> As we phase out the PDF, this page may receive updates not reflected in the static PDF.

In the previous chapter, we showed how to describe an interesting objective function for machine learning, but we need a way to find the optimal $\Theta^* = \arg\min_\Theta J(\Theta)$, particularly when the objective function is not amenable to analytical optimization. For example, this can be the case when $J(\Theta)$ involves a more complex loss function, or more general forms of regularization. It can also be the case when there are simply too many parameters to learn for it to be computationally feasible.

There is an enormous and fascinating literature on the mathematical and algorithmic foundations of optimization, but for this class, we will consider one of the simplest methods, called *gradient descent.*

You might want to consider studying optimization some day! It's one of the fundamental tools enabling machine learning, and it's a beautiful and deep field.

Intuitively, in one or two dimensions, we can easily think of $J(\Theta)$ as defining a surface over Θ; that same idea extends to higher dimensions. Now, our objective is to find the Θ value at the lowest point on that surface. One way to think about gradient descent is that you start at some arbitrary point on the surface, look to see in which direction the "hill" goes down most steeply, take a small step in that direction, determine the direction of steepest descent from where you are, take another small step, etc.

Below, we explicitly give gradient descent algorithms for one and multidimensional objective functions (Section 3.1 and Section 3.2). We then illustrate the application of gradient descent to a loss function which is not merely mean squared loss (Section 3.3). And we present an important method known as *stochastic gradient descent* (Section 3.4), which is especially useful when datasets are too large for descent in a single batch, and has some important behaviors of its own.

3.1 Gradient descent in one dimension

We start by considering gradient descent in one dimension. Assume $\Theta \in \mathbb{R}$, and that we know both $J(\Theta)$ and its first derivative with respect to Θ, $J'(\Theta)$. Here is pseudo-code for gradient descent on an arbitrary function f. Along with f and its gradient $\nabla_\Theta f$ (which, in the case of a scalar Θ, is the same as its derivative f'), we have to specify some hyper-parameters. These hyper-parameters include the initial value for parameter Θ, a *step-size* hyper-parameter η, and an *accuracy* hyper-parameter ϵ.

The hyper-parameter η is often called *learning rate* when gradient descent is applied in machine learning. For simplicity, η may be taken as a constant, as is the case in the pseudo-code below; and we'll see adaptive (non-constant) step-sizes soon. What's important to notice though, is that even when η is constant, the actual magnitude of the change to Θ may not be constant, as that change depends on the magnitude of the gradient itself too.

1: **procedure** 1D-GRADIENT-DESCENT($\Theta_{init}, \eta, f, f', \epsilon$)
2:    $\Theta^{(0)} \leftarrow \Theta_{init}$
3:    $t \leftarrow 0$
4:    **repeat**
5:        $t \leftarrow t + 1$
6:        $\Theta^{(t)} = \Theta^{(t-1)} - \eta f'(\Theta^{(t-1)})$
7:    **until** $|f'(\Theta^{(t)})| < \epsilon$
8:    **return** $\Theta^{(t)}$
9: **end procedure**

Note that this algorithm terminates when the derivative of the function f is sufficiently small. There are many other reasonable ways to decide to terminate, including:

- Stop after a fixed number of iterations T, i.e., when $t = T$. Practically, this is the most common choice.

- Stop when the change in the value of the parameter Θ is sufficiently small, i.e., when $\left|\Theta^{(t)} - \Theta^{(t-1)}\right| < \epsilon$.

> **? Study Question**
>
> Consider all of the potential stopping criteria for `1D-Gradient-Descent`, both in the algorithm as it appears and listed separately later. Can you think of ways that any two of the criteria relate to each other?

Theorem 3.1 *Choose any small distance $\tilde{\epsilon} > 0$. If we assume that f has a minimum, is sufficiently "smooth" and convex, and if the learning rate η is sufficiently small, gradient descent will reach a point within $\tilde{\epsilon}$ of a global optimum point Θ.*

However, we must be careful when choosing the learning rate to prevent slow convergence, non-converging oscillation around the minimum, or divergence.

The following plot illustrates a convex function $f(x) = (x - 2)^2$, starting gradient descent at $x_{init} = 4.0$ with a step-size of $1/2$. It is very well-behaved!
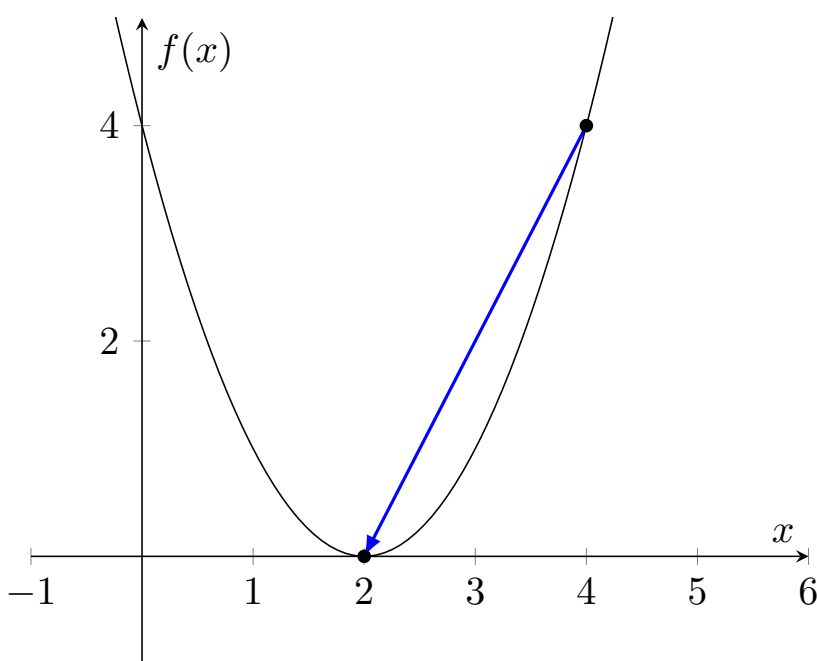
If f is non-convex, where gradient descent converges to depends on x_{init}. First, let's establish some definitions. Let f be a real-valued function defined over some domain D. A point $x_0 \in D$ is called a *global minimum point* of f if $f(x_0) \le f(x)$ for all other $x \in D$. A point $x_0 \in D$ is instead called a *local minimum point* of a function f if there exists some constant $\epsilon > 0$ such that for all x within the interval defined by $d(x, x_0) < \epsilon$, $f(x_0) \le f(x)$, where d is some distance metric, e.g., $d(x, x_0) = ||x - x_0||$. A global minimum point is also a local minimum point, but a local minimum point does not have to be a global minimum point.

> **? Study Question**
>
> What happens in this example with very small η? With very big η?

If f is non-convex (and sufficiently smooth), one expects that gradient descent (run long enough with small enough learning rate) will get very close to a point at which the gradient is zero, though we cannot guarantee that it will converge to a global minimum point.

There are two notable exceptions to this common sense expectation: First, gradient descent can get stagnated while approaching a point x which is not a local minimum or maximum, but satisfies $f'(x) = 0$. For example, for $f(x) = x^3$, starting gradient descent from the initial guess $x_{init} = 1$, while using learning rate $\eta < 1/3$ will lead to $x^{(k)}$ converging to zero as $k \to \infty$. Second, there are functions (even convex ones) with no minimum points, like $f(x) = \exp(-x)$, for which gradient descent with a positive learning rate converges to $+\infty$.

The plot below shows two different x_{init}, and how gradient descent started from each point heads toward two different local optimum points.
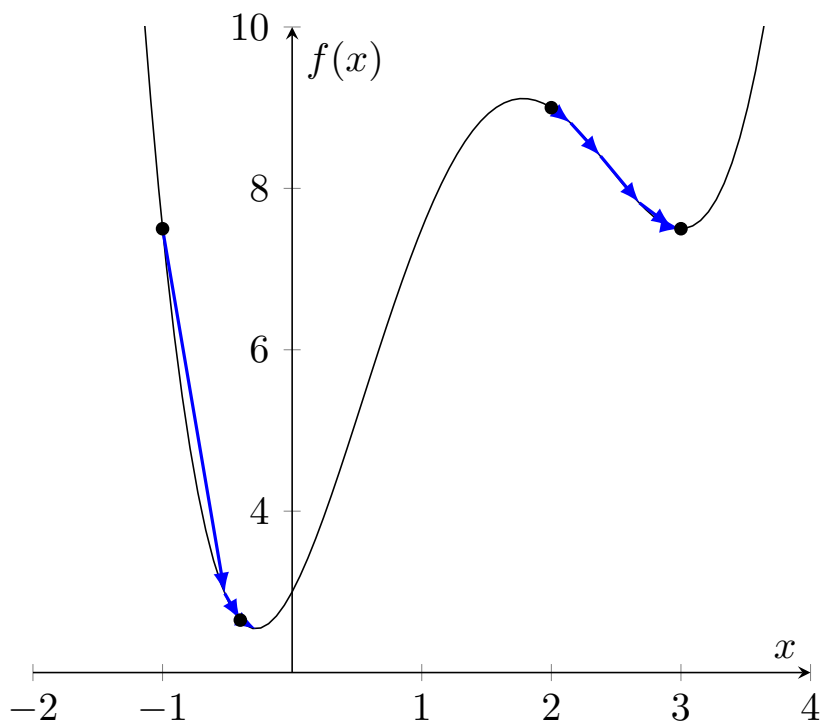
3.2 Multiple dimensions

The extension to the case of multi-dimensional Θ is straightforward. Let's assume $\Theta \in \mathbb{R}^m$, so $f : \mathbb{R}^m \to \mathbb{R}$.

The gradient of f with respect to Θ is

$$\nabla_\Theta f = \begin{bmatrix} \partial f/\partial \Theta_1 \\ \vdots \\ \partial f/\partial \Theta_m \end{bmatrix}$$

The algorithm remains the same, except that the update step in line 5 becomes

$$\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$$

and any termination criteria that depended on the dimensionality of Θ would have to change. The easiest thing is to keep the test in line 6 as $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$, which is sensible no matter the dimensionality of Θ.

> **? Study Question**
>
> Which termination criteria from the 1D case were defined in a way that assumes Θ is one dimensional?

3.3 Application to regression

Recall from the previous chapter that choosing a loss function is the first step in formulating a machine-learning problem as an optimization problem, and for regression we studied the mean square loss, which captures losws as $(\text{guess} - \text{actual})^2$. This leads to the *ordinary least squares* objective

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} - y^{(i)} \right)^2 \ .$$

We use the gradient of the objective with respect to the parameters,

$$\nabla_\theta J = \frac{2}{n} \underbrace{X^T}_{d \times n} \underbrace{(X\theta - Y)}_{n \times 1} \ , \tag{3.1}$$

to obtain an analytical solution to the linear regression problem. Gradient descent could also be applied to numerically compute a solution, using the update rule

$$\theta^{(t)} = \theta^{(t-1)} - \eta \frac{2}{n} \sum_{i=1}^{n} \left(\left[\theta^{(t-1)} \right]^T x^{(i)} - y^{(i)} \right) x^{(i)} \ .$$

3.3.1 Ridge regression

Now, let's add in the regularization term, to get the ridge-regression objective:

$$J_{\text{ridge}}(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} + \theta_0 - y^{(i)} \right)^2 + \lambda \|\theta\|^2 \ .$$

Recall that in ordinary least squares, we finessed handling θ_0 by adding an extra dimension of all 1's. In ridge regression, we really do need to separate the parameter vector θ from the offset θ_0, and so, from the perspective of our general-purpose gradient descent method, our whole parameter set Θ is defined to be $\Theta = (\theta, \theta_0)$. We will go ahead and find the gradients separately for each one:

$$\nabla_\theta J_{\text{ridge}}(\theta, \theta_0) = \frac{2}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} + \theta_0 - y^{(i)} \right) x^{(i)} + 2\lambda\theta$$

$$\frac{\partial J_{\text{ridge}}(\theta, \theta_0)}{\partial \theta_0} = \frac{2}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} + \theta_0 - y^{(i)} \right) \ .$$

Note that $\nabla_\theta J_{\text{ridge}}$ will be of shape $d \times 1$ and $\partial J_{\text{ridge}}/\partial \theta_0$ will be a scalar since we have separated θ_0 from θ here.

? **Study Question**

Convince yourself that the dimensions of all these quantities are correct, under the assumption that θ is $d \times 1$. How does d relate to m as discussed for Θ in the previous section?

? Study Question

Compute $\nabla_\theta \|\theta\|^2$ by finding the vector of partial derivatives $(\partial\|\theta\|^2/\partial\theta_1, \ldots, \partial\|\theta\|^2/\partial\theta_d)$. What is the shape of $\nabla_\theta\|\theta\|^2$?

? Study Question

Compute $\nabla_\theta J_{\text{ridge}}(\theta^T x + \theta_0, y)$ by finding the vector of partial derivatives $(\partial J_{\text{ridge}}(\theta^T x + \theta_0, y)/\partial\theta_1, \ldots, \partial J_{\text{ridge}}(\theta^T x + \theta_0, y)/\partial\theta_d)$.

? Study Question

Use these last two results to verify our derivation above.

Putting everything together, our gradient descent algorithm for ridge regression becomes

Beware double superscripts! $[\theta]^T$ is the transpose of the vector θ.

1: **procedure** RR-GRADIENT-DESCENT$(\theta_{init}, \theta_{0init}, \eta, \epsilon)$
2:   $\theta^{(0)} \leftarrow \theta_{init}$
3:   $\theta_0^{(0)} \leftarrow \theta_{0init}$
4:   $t \leftarrow 0$
5:   **repeat**
6:     $t \leftarrow t + 1$
7:     $\theta^{(t)} = \theta^{(t-1)} - \eta \left(\frac{1}{n} \sum_{i=1}^n \left(\theta^{(t-1)^T} x^{(i)} + \theta_0^{(t-1)} - y^{(i)} \right) x^{(i)} + \lambda\theta^{(t-1)} \right)$
8:     $\theta_0^{(t)} = \theta_0^{(t-1)} - \eta \left(\frac{1}{n} \sum_{i=1}^n \left(\theta^{(t-1)^T} x^{(i)} + \theta_0^{(t-1)} - y^{(i)} \right) \right)$
9:   **until** $\left| J_{\text{ridge}}(\theta^{(t)}, \theta_0^{(t)}) - J_{\text{ridge}}(\theta^{(t-1)}, \theta_0^{(t-1)}) \right| < \epsilon$
10:   **return** $\theta^{(t)}, \theta_0^{(t)}$
11: **end procedure**

? Study Question

Is it okay that λ doesn't appear in line 8?

? Study Question

Is it okay that the 2's from the gradient definitions don't appear in the algorithm?

3.4 Stochastic gradient descent

When the form of the gradient is a sum, rather than take one big(ish) step in the direction of the gradient, we can, instead, randomly select one term of the sum, and take a very small step in that direction. This seems sort of crazy, but remember that all the little steps would average out to the same direction as the big step if you were to stay in one place. Of course, you're not staying in that place, so you move, in expectation, in the direction of the gradient.

Most objective functions in machine learning can end up being written as an average over data points, in which case, stochastic gradient descent (sgd) is implemented by picking a data point randomly out of the data set, computing the gradient as if there were only that one point in the data set, and taking a small step in the negative direction.

Let's assume our objective has the form

$$f(\Theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\Theta) \ ,$$

where n is the number of data points used in the objective (and this may be different from the number of points available in the whole data set).

Here is pseudocode for applying sgd to such an objective f; it assumes we know the form of $\nabla_\Theta f_i$ for all i in $1 \ldots n$:

Sometimes you will see that the objective being written as a sum, instead of an average. In the "sum" convention, the $\frac{1}{n}$ normalizing constant is getting "absorbed" into individual f_i.

$$f(\Theta) = \sum_{i=1}^{n} f_i(\Theta) \ .$$

1: **procedure** STOCHASTIC-GRADIENT-DESCENT$(\Theta_{init}, \eta, f, \nabla_\Theta f_1, \ldots, \nabla_\Theta f_n, T)$
2:     $\Theta^{(0)} \leftarrow \Theta_{init}$
3:     **for** $t \leftarrow 1$ **do**
4:         randomly select $i \in \{1, 2, \ldots, n\}$
5:         $\Theta^{(t)} = \Theta^{(t-1)} - \eta(t) \, \nabla_\Theta f_i(\Theta^{(t-1)})$
6:     **end for**
7: **end procedure**

Note that now instead of a fixed value of η, η is indexed by the iteration of the algorithm, t. Choosing a good stopping criterion can be a little trickier for sgd than traditional gradient descent. Here we've just chosen to stop after a fixed number of iterations T.

For sgd to converge to a local optimum point as t increases, the learning rate has to decrease as a function of time. The next result shows one learning rate sequence that works.

Theorem 3.2 *If f is convex, and $\eta(t)$ is a sequence satisfying*

$$\sum_{t=1}^{\infty} \eta(t) = \infty \ \text{ and } \ \sum_{t=1}^{\infty} \eta(t)^2 < \infty \ ,$$

then SGD converges with probability one* to the optimal Θ.*

Why these two conditions? The intuition is that the first condition, on $\sum \eta(t)$, is needed to allow for the possibility of an unbounded potential range of exploration, while the second condition, on $\sum \eta(t)^2$, ensures that the learning rates get smaller and smaller as t increases.

One "legal" way of setting the learning rate is to make $\eta(t) = 1/t$ but people often use rules that decrease more slowly, and so don't strictly satisfy the criteria for convergence.

> **? Study Question**
>
> If you start a long way from the optimum, would making $\eta(t)$ decrease more slowly tend to make you move more quickly or more slowly to the optimum?

There are multiple intuitions for why sgd might be a better choice algorithmically than regular gd (which is sometimes called *batch* gd (bgd)):

- bgd typically requires computing some quantity over every data point in a data set. sgd may perform well after visiting only some of the data. This behavior can be useful for very large data sets – in runtime and memory savings.

- If your f is actually non-convex, but has many shallow local optimum points that might trap bgd, then taking *samples* from the gradient at some point Θ might "bounce" you around the landscape and away from the local optimum points.

- Sometimes, optimizing f really well is not what we want to do, because it might overfit the training set; so, in fact, although sgd might not get lower training error than bgd, it might result in lower test error.