CHAPTER 6

Gradient Descent

In the last chapter, we have shown how to describe an interesting objective function for machine learning, but we need a way to find the optimal $\Theta^* = \arg\min_\Theta J(\Theta)$. There is an enormous, 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*.

Which you should consider studying some day!

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.

Here's a very old-school humorous description of gradient descent and other optimization algorithms using analogies involving kangaroos: `ftp://ftp.sas.com/pub/neural/kangaroos.txt`

1 One dimension

We will 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 f', we have to specify the initial value for parameter Θ, a *step-size* parameter η, and an *accuracy* parameter ϵ:

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

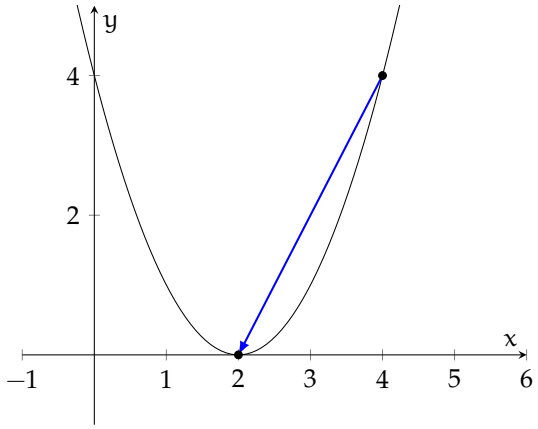
Note that there are many other reasonable ways to decide to terminate, including when $\left|\Theta^{(t)} - \Theta^{(t-1)}\right| < \epsilon$ or when $\left|f(\Theta^{(t)}) - f(\Theta^{(t-1)})\right| < \epsilon$.

Theorem 1.1. *If J is convex, for any desired accuracy ϵ, there is some step size η such that gradient descent will converge to within ϵ of the optimal Θ.*

A function is convex if the line segment between any two points on the graph of the function lies above or on the graph.

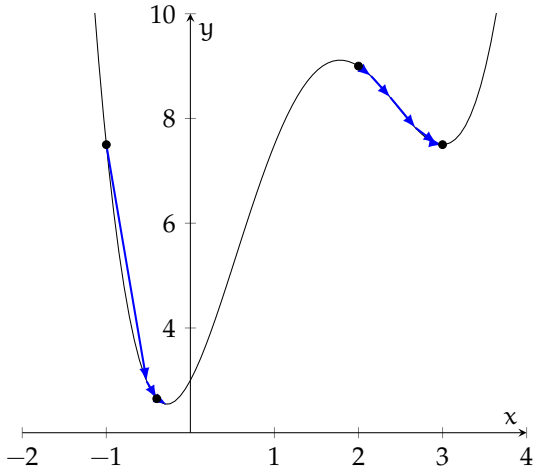
However, we must be careful when choosing the step size to prevent slow convergence, oscillation around the minimum, or divergence.

The following plot illustrates a convex function $f(x) = (x-2)^2$, starting gradient descent at $\theta_{\texttt{init}} = 4.0$ with a step-size of $1/2$. It is very well-behaved!



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

If J is non-convex, where gradient descent converges to depends on θ_{init}. When it reaches a value of θ where $f'(\theta) = 0$ and $f''(\theta) > 0$, but it is not a minimum of the function, it is called a *local minimum* or *local optimum*.



2 Multiple dimensions

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

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

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

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

and we have to change the termination criterion. The easiest thing is to replace the test in line 6 with $\left|f(\Theta^{(t)}) - f(\Theta^{(t-1)})\right| < \epsilon$, which is sensible no matter the dimensionality of Θ.

3 Application to SVM objective

There are two slight "wrinkles" involved in applying gradient descent to the SVM objective.

We begin by stating the objective and the gradient necessary for doing gradient descent. In our problem where we are considering linear separators, the entire parameter vector is described by parameter vector θ and scalar θ_0 and so we will have to adjust them both and compute gradients of J with respect to each of them. The objective and gradient (note that we have replaced the constant λ with $\frac{\lambda}{2}$ for convenience), are

The following step requires passing familiarity with matrix derivatives. A foolproof way of computing them is to compute partial derivative of J with respect to each component θ_i of θ.

$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L_h\left(y^{(i)}(\theta^T x^{(i)} + \theta_0)\right) + \frac{\lambda}{2}\|\theta\|^2$$

$$\nabla_\theta J = \frac{1}{n}\sum_{i=1}^{n} L_h'\left(y^{(i)}(\theta^T x^{(i)} + \theta_0)\right) y^{(i)} x^{(i)} + \lambda\theta$$

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{n}\sum_{i=1}^{n} L_h'\left(y^{(i)}(\theta^T x^{(i)} + \theta_0)\right) y^{(i)}.$$

Note that $\nabla_\theta J$ will be of shape $d \times 1$ and $\frac{\partial J}{\partial \theta_0}$ 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 $\nabla_\theta \left(y(\theta^T x + \theta_0)\right)$ by finding the vector of partial derivatives $(\partial \left(y(\theta^T x + \theta_0)\right) / \partial\theta_1, \ldots, \partial \left(y(\theta^T x + \theta_0)\right) / \partial\theta_d)$.

Study Question: Use these last two results to verify our derivation above.

Recall the hinge-loss

$$L_h(v) = \begin{cases} 1 - v & \text{if } v < 1 \\ 0 & \text{otherwise} \end{cases}.$$

This loss is not differentiable, since the derivative at $v = 1$ doesn't exist! So we consider the subgradient

You don't have to really understand the idea of a subgradient, just that it has value 0 at $v = 1$ here.

$$L_h'(v) = \begin{cases} -1 & \text{if } v < 1 \\ 0 & \text{otherwise} \end{cases}.$$

This gives us a complete definition of $\nabla_\theta J$ and $\partial J / \partial \theta_0$. Finally, our gradient descent algorithm becomes

SVM-GRADIENT-DESCENT($\theta_{init}, \theta_{0init}, \eta, J, \epsilon$)

1 $\theta^{(0)} = \theta_{init}$
2 $\theta_0^{(0)} = \theta_{0init}$
3 $t = 0$
4 **repeat**
5 $\quad t = t + 1$
6 $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \left(\frac{1}{n} \sum_{i=1}^{n} \left\{ \begin{array}{l} -1 \text{ if } y^{(i)} \left(\theta^{(t-1)T} x^{(i)} + \theta_0^{(t-1)} \right) < 1 \\ 0 \text{ otherwise} \end{array} \right\} y^{(i)} x^{(i)} + \lambda \theta^{(t-1)} \right)$
7 $\quad \theta_0^{(t)} = \theta_0^{(t-1)} - \eta \left(\frac{1}{n} \sum_{i=1}^{n} \left\{ \begin{array}{l} -1 \text{ if } y^{(i)} \left(\theta^{(t-1)T} x^{(i)} + \theta_0^{(t-1)} \right) < 1 \\ 0 \text{ otherwise} \end{array} \right\} y^{(i)} \right)$
8 **until** $\left| J(\theta^{(t)}, \theta_0^{(t)}) - J(\theta^{(t-1)}, \theta_0^{(t-1)}) \right| < \epsilon$
9 **return** $\theta^{(t)}, \theta_0^{(t)}$

Study Question: Is it okay that λ doesn't appear in line 7?

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.

The word "stochastic" means probabilistic, or random; so does "aleatoric," which is a very cool word. Look up aleatoric music, sometime.

Most objective functions in machine learning can end up being written as a sum 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

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

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

STOCHASTIC-GRADIENT-DESCENT($\Theta_{init}, \eta, f, \nabla_\Theta f_1, \ldots, \nabla_\Theta f_n, T$)

1 $\Theta^{(0)} = \Theta_{init}$
2 **for** $t = 1$ **to** T
3 $\quad$ randomly select $i \in \{1, 2, \ldots, n\}$
4 $\quad \Theta^{(t)} = \Theta^{(t-1)} - \eta(t) \nabla_\Theta f_i(\Theta^{(t-1)})$
5 **return** $\Theta^{(t)}$

Note that now instead of a fixed value of η, it is indexed by the iteration of the algorithm, t. For SGD to converge to a local optimum as t increases, the step size has to decrease as a function of time.

Theorem 4.1. *If J is convex, and $\eta(t)$ is a sequence satisfying*

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

then SGD converges almost surely *to the optimal* Θ.

We have left out some gnarly conditions in this theorem, and note that *almost surely* is a technical term from probability theory, not a lack of courage on the part of the author.

One "legal" way of setting the step size 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 two intuitions for why SGD might be a better choice algorithmically than regular GD (which is sometimes called *batch* GD (BGD)):

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