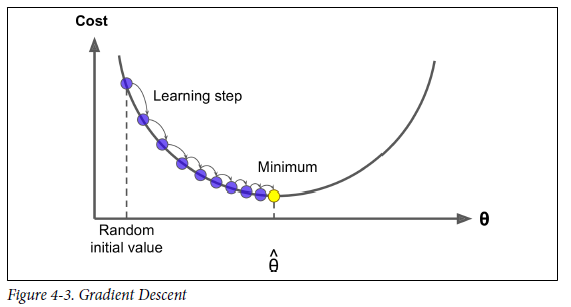
**Gradient Decent:**

Frequently when doing data science, we’ll be trying to the find the best model for a certain situation. And usually “best” will mean something like “minimizes the error of the model” or “maximizes the likelihood of the data.” In other words, it will represent the solution to some sort of optimization problem. This means we’ll need to solve a number of optimization problems. And in particular, we’ll need to solve them from scratch. Our approach will be a technique called gradient descent, which lends itself pretty well to a from-scratch treatment.

Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

**You can either define a utility function (or fitness function) that measures how good your model is, or you can define a cost function that measures how bad it is.**

Using an iterative optimization approach, called Gradient Descent (GD) that gradually tweaks the model parameters to minimize the cost function over the training set.



Concretely, you start by filling θ with random values (this is called random initialization), and then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function until the algorithm converges to a minimum.

**When using Gradient Descent, you should ensure that all features have a similar scale (e.g., using Scikit-Learn’s StandardScaler class), or else it will take much longer to converge. Caution: Check with boosting. I don’t know if it true or not.**

**Learning Rate**

An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time. On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution. Finally, not all cost functions look like nice regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult. Figure 4-6 shows the two main challenges with Gradient Descent: if the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum. If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.

