Gradient Descent and Linear Regression

Notation

We will be using a lot of index in this lecture so let me spell it out for you. We need two indices: one for telling us which data points and the other for telling which feature I'm talking about. The data points will be denote with upper index in parenthesis. Specifically $\mathbf{x}^{(1)}$ means the vector feature of the first data point, $\vec{x}^{(2)}$ means the feature vector of the second data point and so on. If I want to refer to specific feature I'll be using the lower index. Specifically, $x_1^{(2)}$ means the first feature of the second data point.

In summary, $x_j^{(i)}$ is the j-th feature of the i-th data point.

Linear Regression

Suppose you have the following dataset of the price of a condo and the area. See exercise.

It's clear that you can predict the value of the new data point by just fitting a straight line to the dataset then use the straight line to predict the price of the new condo.

This can be done simply by finding the line that minimize the sum of the distance between the point and the line squared.

$$\text{cost}(h_{\mathbf{w}}) = \frac{1}{2N}\sum_{i=1}^{N}\left[h_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right]^2$$

where

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

To understand this expression, first, the cost is a function of our hypothesis $h_{\mathbf{w}}$(line) which is our guess of the true model. More specifically, since the hypothesis is parametrize by the weight the cost function is just a function of weights, $\mathbf{w}$. Each hypothesis/line has associated cost and we construct the cost function such that the "best" line is the line that minimize this cost function. Try drawing a few lines and you will see this align with our rough idea of "best" line. The factor of $\frac{1}{2N}$ is just a constant convention; you can ignore it.

The main ingredient is the expression inside the sum over all the data points: each term inside the sum over the data points is the difference between the predicted from our hypothesis($h_{\mathbf{w}}(\mathbf{x}^{(i)})$) value and the true value $y^{(i)}$. Then we square it so that the negative contribution doesn't cancel with positive contribution. The higher the value the further away the line from the point.

The hypothesis, $h_{\mathbf{w}}(\mathbf{x})$, is simply just the linear combination of feature vector,$\mathbf{x}$. Note also that here we use the convention that $x_0 = 1$. This represents a straight line in 2D and a plane in 3D and hyper plane in higher dimensional data similar to the perceptron. Each $\mathbf{w}$ corresponds to each line.

So all we need to do now is to find $\mathbf{w}$ that minimize the cost function. There is actually a closed form for the solution which is more efficient that what we are going to do here. Those of you taking numerical method before "should" be able to derive the formula from first principle but in this class we are going to use (less efficient) gradient descent to do it.

Gradient Descent

So we have reduced the problem for fitting a line to the problem of just finding parameters,$\mathbf{w}$, that minimize the cost function. This is a very common task in pattern recognition. We need to pick the best hypothesis,g, from hypothesis set $\mathcal{H}$. The "best" one has to be special in some way: it has to minimize something.

The gradient descent algorithm is the following:

1. Start at some location.
2. Look around and walk in the direction which decrease the function the most. This happens to be the gradient, $-\nabla f$

   $$\mathbf{w}' = \mathbf{w} + \text{step size} \times \left(\frac{-\nabla f}{\|\nabla f\|} \right) \tag{1}$$

   You may be confused with the gradient. The cost function depends only on the weight, the gradient that we are trying to find is of those derivative with respect to each w_i.
3. Keep doing the above until all direction around you only make the function value greater.

See the board for 2D example.

There is a question of what we should pick for the step size. We need two things.

- We want the step size to be large when we are far away from the minimum. This way we can get the minimum quicker.
- We want the step size to be small when we are closed to the minimum. This way we won't step over the minimum and we got a good accuracy.

This is sort of recursive question since we want a function that sort of know where the minimum is to pick the value. However, this is actually an easy thing to do: $\|\nabla f\|$ does what we need. Near the minimum gradient is small(zero small). So we can establish that step size $\propto \|\nabla f\|$. In other words,

$$\text{step size} = \eta \|\nabla f\|,$$

where η is called the learning rate does exactly what we need. The learning rate is typically less than one: 0.01 is a common value. If we plug this into Equation 1, we get a very nice equation:

$$\mathbf{w}' = \mathbf{w} - \eta \nabla f \tag{2}$$

Let me summarize what to do for gradient descent:

1. Start with some $\mathbf{w}$
2. Update $\mathbf{w}$ with

   $$\mathbf{w}' = \mathbf{w} - \eta \nabla f \tag{3}$$

   Or simultaneously update

   $$w_i' = w_i - \eta \frac{\partial}{\partial w_i} f \tag{4}$$

   It is important to note that the gradient must be computed with old $\mathbf{w}$
3. Stop when the the ∇f is low

This algorithm will be used a lot in this course. You should write your own for once then for the rest just use the package to do this for you. It is simple but the bad news is that this method is susceptible to local minima. This is something to keep in mind when you apply gradient descent to real world application. Fortunately, for the problem of linear regression we are dealing with, the cost function is convex function which guarantees that there is no local minima.

One can play with the learning rate to get a reasonably fast and accurate convergence. High η is susceptible to going over the minima and may not even converge, but if it does, it will converge very fast. Low η is less susceptible to divergent but it converge slowly. Typical value used is 0.1, 0.01 etc.

Linear Regression II

Now we are equipped with the tools necessary to find the best line to fit it. Let me remind you that this not usually not the way people normally do linear regression but it is a very nice illustration for gradient descent.

So, we need to use Equation 4 to update our weight at each iteration. We need to compute the gradient. We have two choice of doing this: numerically or analytically. For this problem, it is easy enough to do it analytically. So, let us do it.

$$\begin{aligned}\nabla \text{cost} &= \nabla \frac{1}{2N}\sum_{i=1}^{N}\left[h_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right]^2 \\ &= \frac{1}{2N}\sum_{i=1}^{N}\nabla\left[h_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right]^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[h_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right] \times \nabla h_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[h_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right] \times \mathbf{x}^{(i)} \end{aligned} \tag{5}$$

The notation may look a bit confusing putting so let me put back in the index for the gradient explicitly

$$\frac{\partial}{\partial w_j}\text{cost} = \frac{1}{N}\sum_{i=1}^{N}\left[h_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right] \times x_j^{(i)} \tag{6}$$

Thus, the update formula for the **w** is

$$w_j' = w_j - \eta\frac{\partial}{\partial w_j}\text{cost} \tag{7}$$

Let me repeat again that the gradient must be evaluated at old weight. This means that you should not overwrite **w** while computing the gradient.

Take Home Lessons

For this lesson I need you to get the big picture. The fact that you define some sort of cost function such that the minimization of the cost function will give you the hypothesis. Plus, the problem of minimization can be solved with gradient descent algorithm. You will be doing this over and over in this course: build a cost function and minimize it.