Stochastic Gradient Decent

Authors: Eliesha Lai, Shruti Kolhatkar, Chirag Ramesh. (PDF)

Introduction

The lecture covered Stochastic Gradient Decent, also it gave brief about the cost function and Gradient Decent. SGD mostly talks about handling the data little bit at same time and deal with it as it comes that is with respect to the time, while explaining this, professor also gave example about the stock market, where we have to constantly adapt with the new data. If we are handling whole data simultaneously the process becomes slow and time consuming, so SGD allows us to work faster and it reacts faster to input and output changes.
##Cost Function

A cost finction, also called loss function is used to measure how well a neural networks predicts the outputs on the test set. One common way we used is mean squared error, in which we measure the differences between the actual value of y and the predicted value of y. It's fair to say that our goal of models is to minimize the cost function.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Figure 1: Imgur

J(θ_0,θ_1) is the cost finction we try to minimize.

So we use the function J(θ_0,θ_1) to minimize the total difference between all predictions and reality by inputting different θ_0 and θ_1 to find the best θ_0 and θ_1.

$h_\theta(x)$ refers to the predicted value, when we input different values of x after a fixed θ is given. Therefore, $h_\theta(x)$ is the "prediction", and the y value is the "reality".

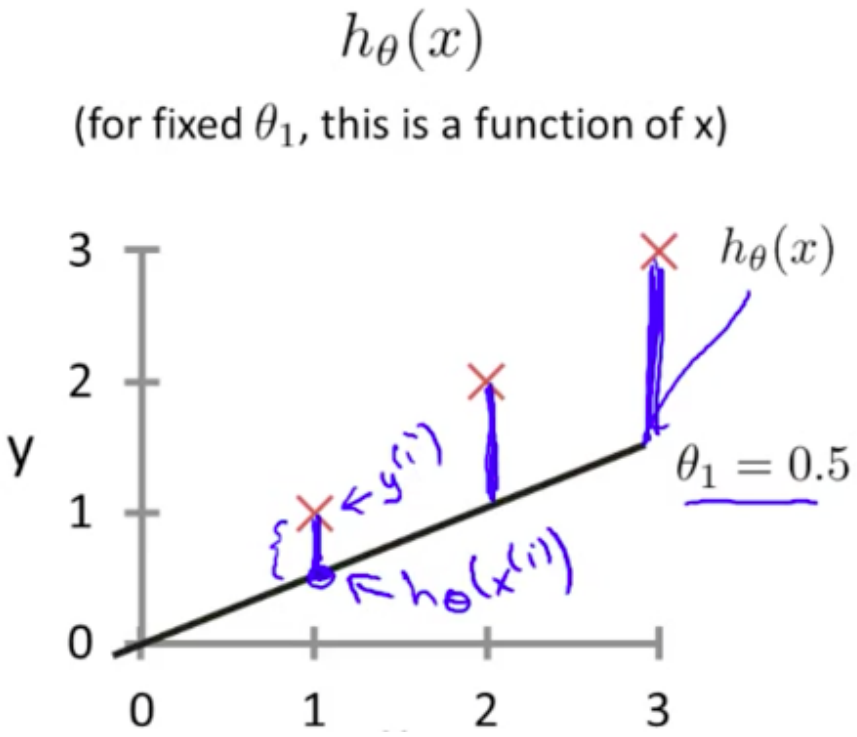


Figure 2: Imgur

The red X is the information we are given (reality), and the black line represents the corresponding values of x and y contained in our function $h_\theta(x)$.In order to make a more accurate prediction, we have to try to make the blue lines (the gap) close to our data set.

Gradient Descent

What is Gradient descent?

Gradient descent is defined as a first-order optimization algorithm that is used for finding a local minimum of a differentiable function. The algorithm minimizes the function by iterating and moving in the direction of the steepest descent.

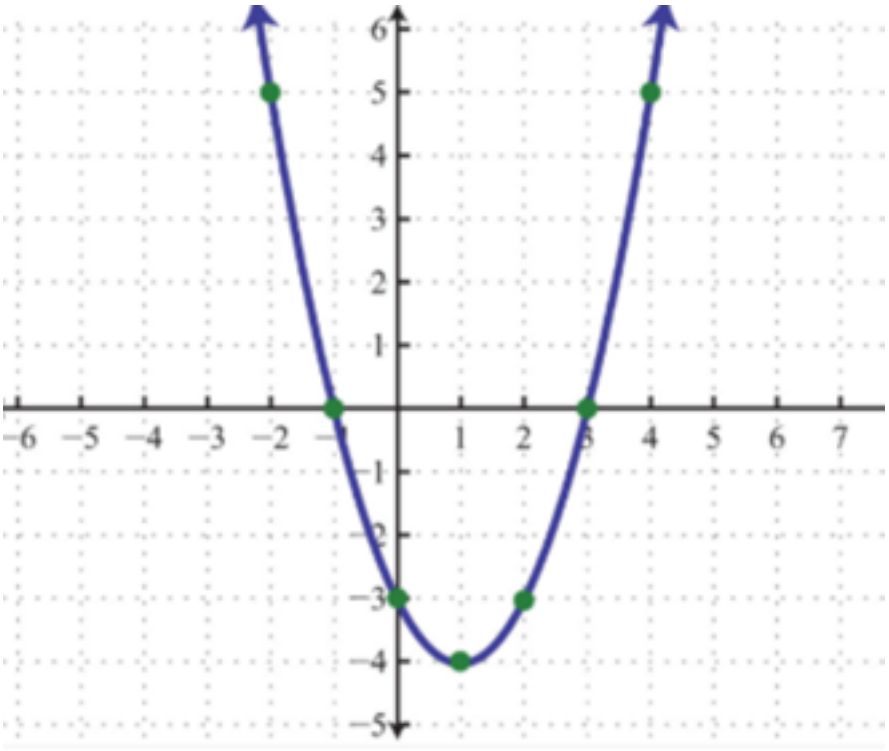


Figure 3: Imgur

In the example above we are basically using the gradient descent algorithm to find the value of “x” where “y” reaches its minimum point. Gradient descent is an iterative algorithm that starts from a random point and travels down the slope until it reaches the lowest point of that function.

The picture below gives us an accurate overview of gradient descent.

Figure 4: Imgur

What Are Gradients?

A gradient is just a way of quantifying the relationship in neural network, which can be graphed as a slope, between error and the weights. The steepness of the slope represents how fast the model is learning.

A steeper slope means large reductions in error are being made and the model is learning fast, whereas if the slope is zero the model is on a plateau and isn't learning. We can move down the slope towards less error by calculating a gradient, a direction of movement (change in the parameters of the network) for our model.

Let's imagine a series of hills and valleys. We want to get to the bottom of the hill (lowest loss). When we start at the top of the hill we can take large steps down the hill and be confident that we are heading towards the lowest point in the valley.

However, as we get closer to the lowest point in the valley, our steps will need to become smaller, or else we could overshoot the true lowest point. Similarly, it's possible that when adjusting the weights of the network, the adjustments can actually take it further away from the point of lowest loss, and therefore the adjustments must get smaller over time.

Now we know that gradients are instructions that tell us which direction to move in (which coefficients should be updated) and how large the steps we should

take are (how much the coefficients should be updated), we can explore how the gradient is calculated.

##Calculating Gradients & Gradient Descent

Gradient descent starts at a place of high loss and by through multiple iterations, takes steps in the direction of lowest loss, aiming to find the optimal weight configuration.

In order to calculate the gradient, we need to know the loss/cost function. We'll use the cost function to determine the derivative.If we represent the loss function as "f", then we can state that the equation for calculating the loss is as follows (we're just running the coefficients through our chosen cost function):

Loss = f(coefficient)

We then calculate the derivative, or determine the slope. Getting the derivative of the loss will tell us which direction is up or down the slope, by giving us the appropriate sign to adjust our coefficients by. We'll represent the appropriate direction as "delta".

delta = derivative_function(loss)

We've now determined which direction is downhill towards the point of lowest loss. This means we can update the coefficients in the neural network parameters and hopefully reduce the loss. How we update the coefficients is shown below. The argument that controls the size of the update is called the "learning rate".

coefficient = coefficient – (learning rate * delta)

We then just repeat this process until the network has converged around the point of lowest loss, which should be near zero.

It's very important to choose the right value for the learning rate. The chosen learning rate must be neither too small or too large. If the step sizes are too large, the network's performance will continue to bounce. In contrast, if the learning rate is too small the network can potentially take an extraordinarily long time to converge on the optimal weights.

The steps of the algorithm are 1. Find the slope of the objective function with respect to each parameter/feature. In other words, compute the gradient of the function.

2. Pick a random initial value for the parameters. (To clarify, in the parabola example, differentiate "y" with respect to "x". If we had more features like x1, x2 etc., we take the partial derivative of "y" with respect to each of the features.)
3. Update the gradient function by plugging in the parameter values.
4. Calculate the step sizes for each feature as : step size = gradient * learning rate.

5. Calculate the new parameters as : new params = old params -step size
6. Repeat steps 3 to 5 until gradient is almost 0.

Stochastic Gradient Descent

Stochastic gradient descent is an iterative method for optimizing an objective function with suitable smoothness properties . It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in high-dimensional optimization problems this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate.Stochastic gradient descent is commonly used practical algorithms for large scale stochastic optimization. It is mainly used where we are getting data continuously, e.g. video streaming.Practically, computing the cost and gradient for the entire training set, can be very slow and sometimes intractable on a single machine if the dataset is too big to fit in main memory.Thus, SGD can be used if we have storage limitations.

The standard gradient descent algorithm updates the parameters θ of the objective J(θ) as,

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)]$$

Figure 5: Imgur

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set. Stochastic Gradient Descent (SGD) simply does away with the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by,

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

Figure 6: Imgur

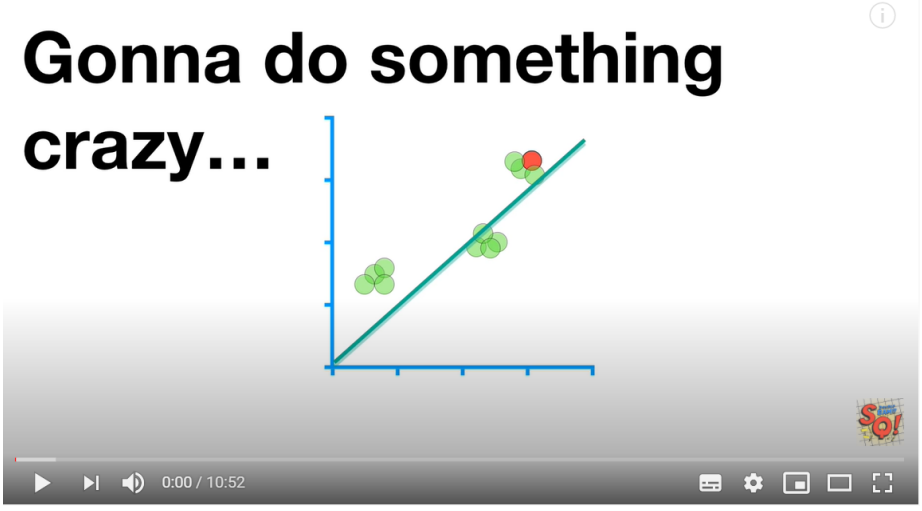
In SGD the learning rate α is typically much smaller than a corresponding learning rate in batch gradient descent because there is much more variance in

the update. Choosing the proper learning rate and schedule (i.e. changing the value of the learning rate as learning progresses) can be fairly difficult. One standard method that works well in practice is to use a small enough constant learning rate that gives stable convergence in the initial epoch (full pass through the training set) or two of training and then halve the value of the learning rate as convergence slows down. An even better approach is to evaluate a held out set after each epoch and anneal the learning rate when the change in objective between epochs is below a small threshold. This tends to give good convergence to a local optima.Generally, the outer loop will iterate over epochs, and for each epoch, we iterates over the dataset in batches as inner loop.

Resources

Gradient Descent:

Stochastic Gradient Descent:



Writing a training loop from scratch: https://keras.io/guides/writing_a_training_loop_from_scratch/