**Task – 6**

**Name : Prasad Nagineni**

**Registration ID: SIRSS1237**

**Q1. Calculate/ derive the gradients used to update the parameters in cost function**

**optimization for simple linear regression**.

Cost function is used to gauge the performance of the machine learning model. Cost function helps to finalize the how well a machine learning model performs. A cost functions basically used to compare the predict values with the actual values.

Linear Regression is a linear model, A model assumes a linear relationship between input variables(X) and the single output variable (Y), Specifically y calculated from linear combination of input variables.

When there is single input variable, the method is referred to as **Simple Linear Regression**.

When there is multiple input variables, the method is referred to as **Multiple Linear Regression**.

In Simple Linear Regression problem (single x, single y), the form of model would be:

y=B0+B1\*x

The coefficients used in simple linear regression can be found using stochastic gradient descent. Linear regression does provide useful exercise for learning stochastic gradient descent which is an important algorithm used to minimize cost functions by Machine Learning algorithm.

Gradient Descent iteration #1:

Let’s consider both coefficients as B0 = 0.0 and B1 = 0.0

Y=0.0 +0.0\*x

We can calculate the error for prediction as follows:

error = p(i) – y(i)

Where

p(i) – Prediction for the i’th instance in our dataset

y(i) – Output variable for i’th instance in the dataset

Now we can calculate the predicted values for y using our starting point coefficients for the first training instance:

x=1, y=1

p(i) = 0+0\*1

p(i) =0

Using the predicted values, we can calculate the error:

error = 0-1

error= -1

We can now use this error in our equation for gradient descent to update the weights. We will start with updating the intercept first.

We can say that B0 is accountable for all of the error. This is to say that updating the weight will use just the error as the gradient. We can calculate the update for the B0 coefficient as follows:

B0(t+1) = B0(t) – alpha\*error

Where

B0(t+1) is updated version of the coefficient we will use on the next training instance.

B0(t) is the current value for B0

alpha is our learning rate

error is the error we calculate for the training instance.

Let’s use a small learning rate of 0.01 and plug the values into the equation to work out what the new and slightly optimized value of B0 will be:

B0(t+1) = 0.0 - 0.01\*-1.0

B0(t+1) = 0.01

Now, let’s look at updating the value for B1. We use the same equation with one small change. The error is filtered by the input that caused it. We can update B1 using the equation:

B1(t+1) = B1(t) – alpha\*error\*x

Where

B1(t+1) is the update coefficient.

B1(t) is the current version of the coefficient.

alpha is the same learning rate described above.

error is the same error calculated above and x is the input value.

Put the values in equation and calculate B1:

B1(t+1) = 0.0 – 0.01 \* -1 \* 1

B1(t+1) = 0.01

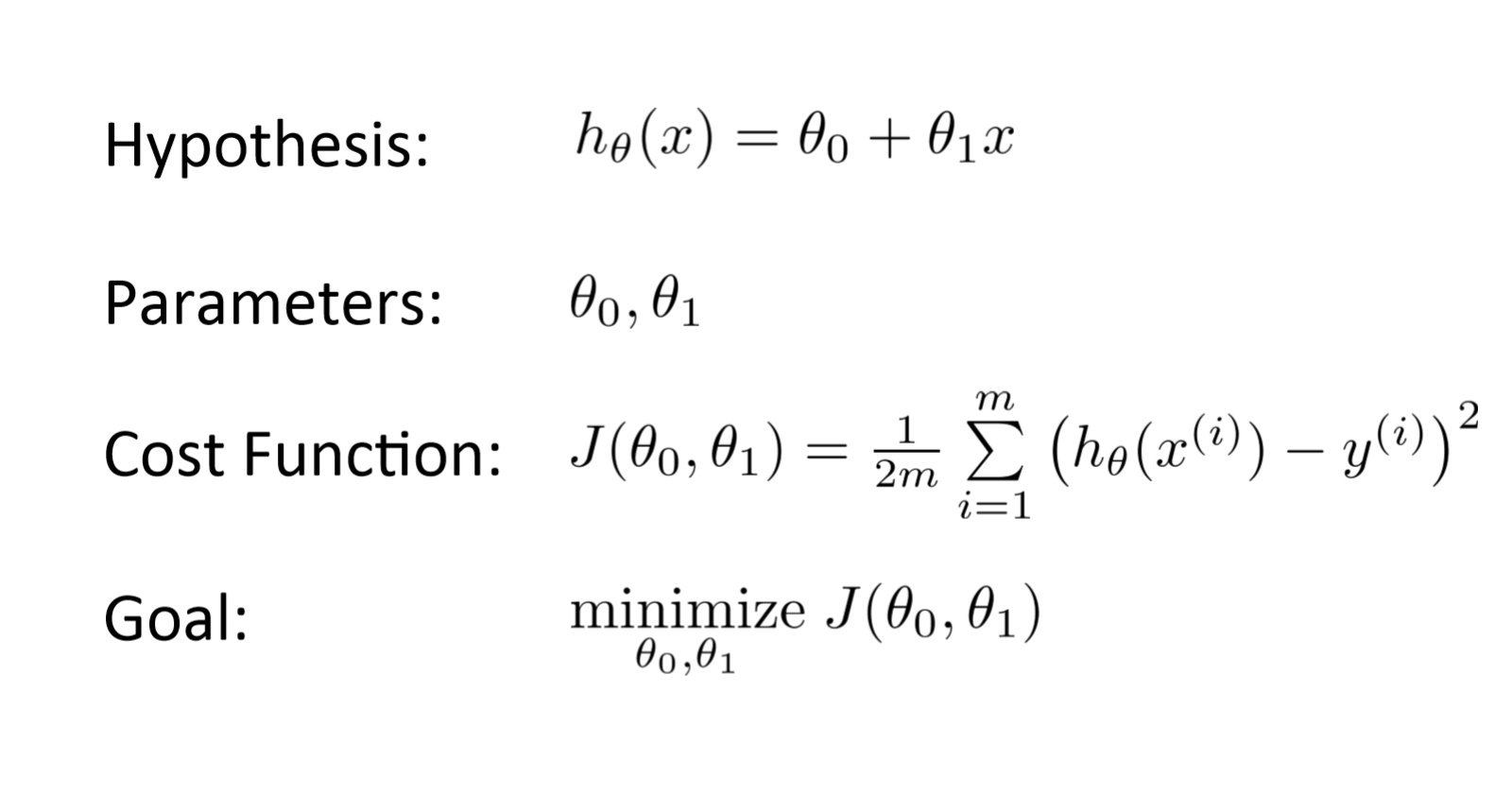
The whole above process is to how to update the coefficients in simple linear regression. After first iteration the weights are updated to B0 = 0.01 and B1 =0.01.

**Q2. What does the sign of gradient say about the relationship between the parameters and cost function?**

A cost function is a measure of the error in prediction committed by an algorithm. It indicates the difference between the predicted and the actual values for a given dataset. Closer the predicted value to the actual value, the smaller the difference and lower the value of the cost function. Lower the value of the cost function, the better the predictive capability of the model. An ideal value of the cost function is zero. Some of the popular cost functions used in machine learning for applications such as regression, classification.

Function that measures the performance of a model for any given data. Cost Function quantifies the error between predicted values and expected values and presents it in the form of a single real number.

Gradient descent is an iterative optimization algorithm for finding the local minimum of a function.



The goal of the gradient descent algorithm is to minimize the given function (cost function).

The sign of the gradient will decide the graph structure and its moving direction. Below are the two point what if gradient is positive and negative.

1. We will reach Local minimum, if we follow proportional to the negative of the gradient which means move away from the gradient of the function at the current point.
2. We will reach Local maximum if we follow proportional to the positive of the gradient which means moving towards gradient of the function.

**Q3. Why Mean squared error is taken as the cost function for regression problems.**

A Cost function is used to gauge the performance of the Machine Learning model. A Machine Learning model devoid of the Cost function is futile. Cost Function helps to analyze how well a Machine Learning model performs. A Cost function basically compares the predicted values with the actual values.

Mean Squared Error (MSE) is the mean squared difference between the actual and predicted values. MSE penalizes high errors caused by outliers by squaring the errors. The optimization algorithms benefit from penalization as it is helpful to find the optimal values for parameters.

Reasons why we use MSE:

1. MSE preferred due to the equation is differentiable.
2. MSE is more sensitive to Outliers.

**Q4. What is the effect of learning rate on optimization, discuss all the cases.**

**Learning rate**

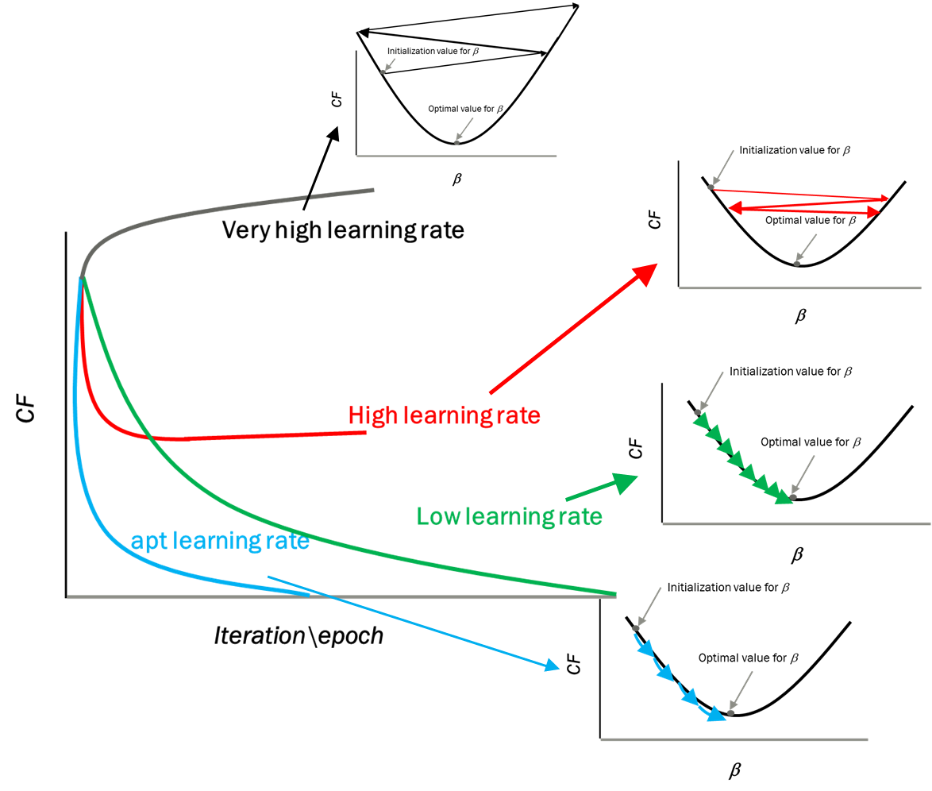
In machine learning, we deal with two types of parameters; 1) machine learnable parameters and 2) hyper-parameters. The Machine learnable parameters are the one which the algorithms learn/estimate on their own during the training for a given dataset. In below equation β0, β1 and β2 are the machine learnable parameters. The Hyper-parameters are the one which the machine learning engineers or data scientists will assign specific values to, to control the way the algorithms learn and also to tune the performance of the model. Learning rate, generally represented by the symbol ‘α’, shown in equation-4, is a hyper-parameter used to control the rate at which an algorithm updates the parameter estimates or learns the values of the parameters.

learning rate

**Effect of different values for learning rate**

Learning rate is used to scale the magnitude of parameter updates during gradient descent. The choice of the value for learning rate can impact two things: 1) how fast the algorithm learns and 2) whether the cost function is minimized or not.

Cost function minimization for different learning rates It can be seen that for an optimal value of the learning rate, the cost function value is minimized in a few iterations. This is represented by the blue line in the figure. If the learning rate used is lower than the optimal value, the number of iterations/epochs required to minimize the cost function is high (takes longer time). This is represented by the green line in the figure. If the learning rate is high, the cost function could saturate at a value higher than the minimum value. This is represented by the red line in the figure. If the learning rate selected is very high, the cost function could continue to increase with iterations/epochs. An optimal learning rate is not easy to find for a given problem. Though getting the right learning is always a challenge, there are some well-researched methods documented to figure out optimal learning rates. Some of these techniques are discussed in the following sections. In all these techniques the fundamental idea is to vary the learning rate dynamically instead of using a constant learning rate.

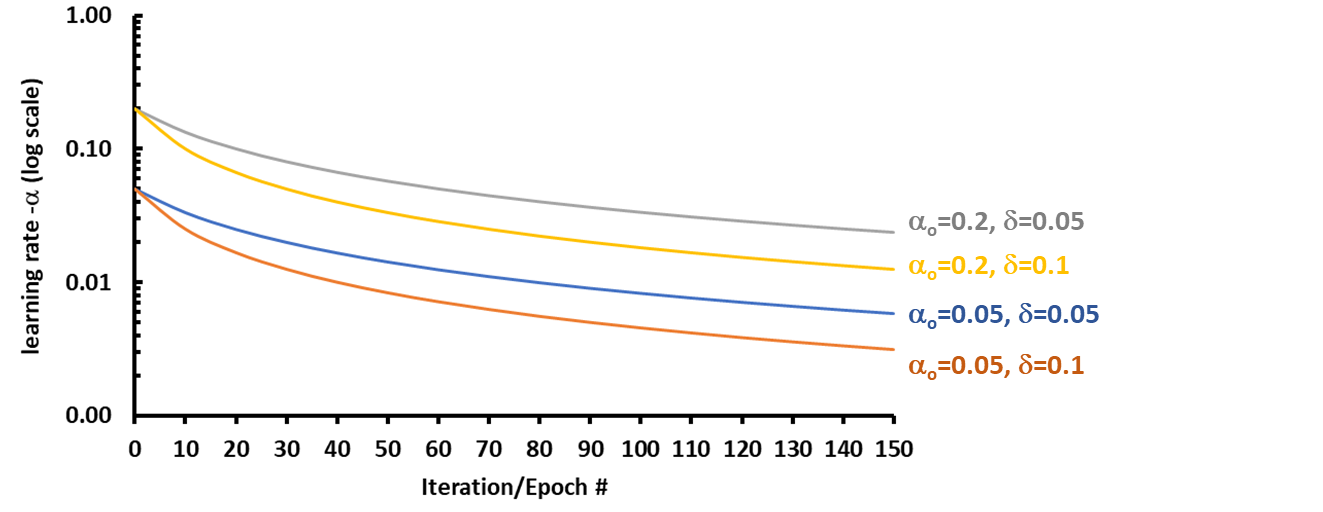


**Decaying Learning rate**

In the decaying learning rate approach, the learning rate decreases with increase in epochs/iterations. The formula used for the decaying learning rate is shown below.

learning rate

In the above equation, o is the initial learning rate, is the decay rate and is the learning rate at a given Epoch number. Figure below shows the learning rate decay with the epoch number for different initial learning rates and decay rates.

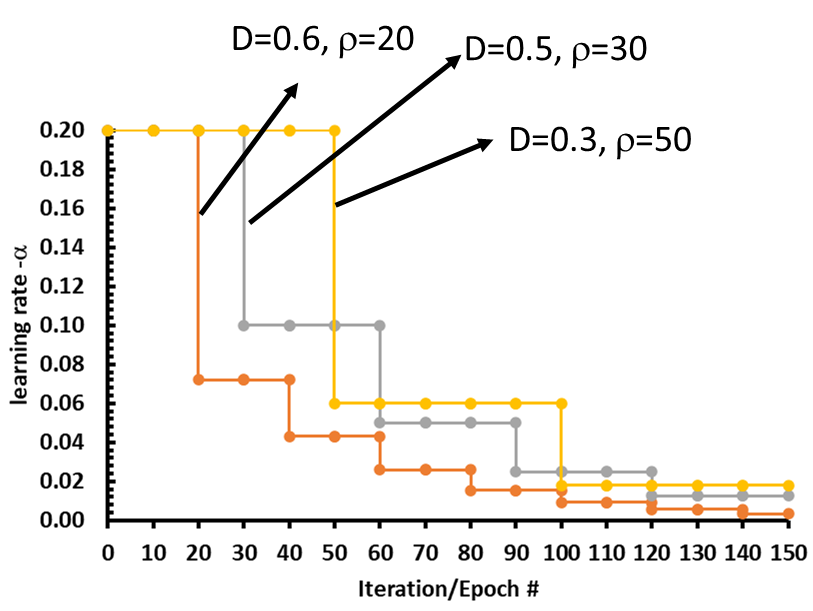


**Scheduled Drop Learning rate**

Unlike the decay method, where the learning rate drops monotonously, in the drop learning rate method, the learning rate is dropped by a predetermined proportion at a predetermined frequency. The formula used to calculate the learning rate for a given epoch is shown in the below equation.

learning rate

In the above equation, o is the initial learning rate, ‘n’ is the epoch/iteration number, ‘D’ is a hyper-parameter which specifies by how much the learning rate has to drop, and ρ is another hyper-parameter which specifies the epoch-based frequency of dropping the learning rate. Figure below shows the learning rate variation with epochs for different values of ‘D’ and ‘ρ’.



Adaptive Learning rate

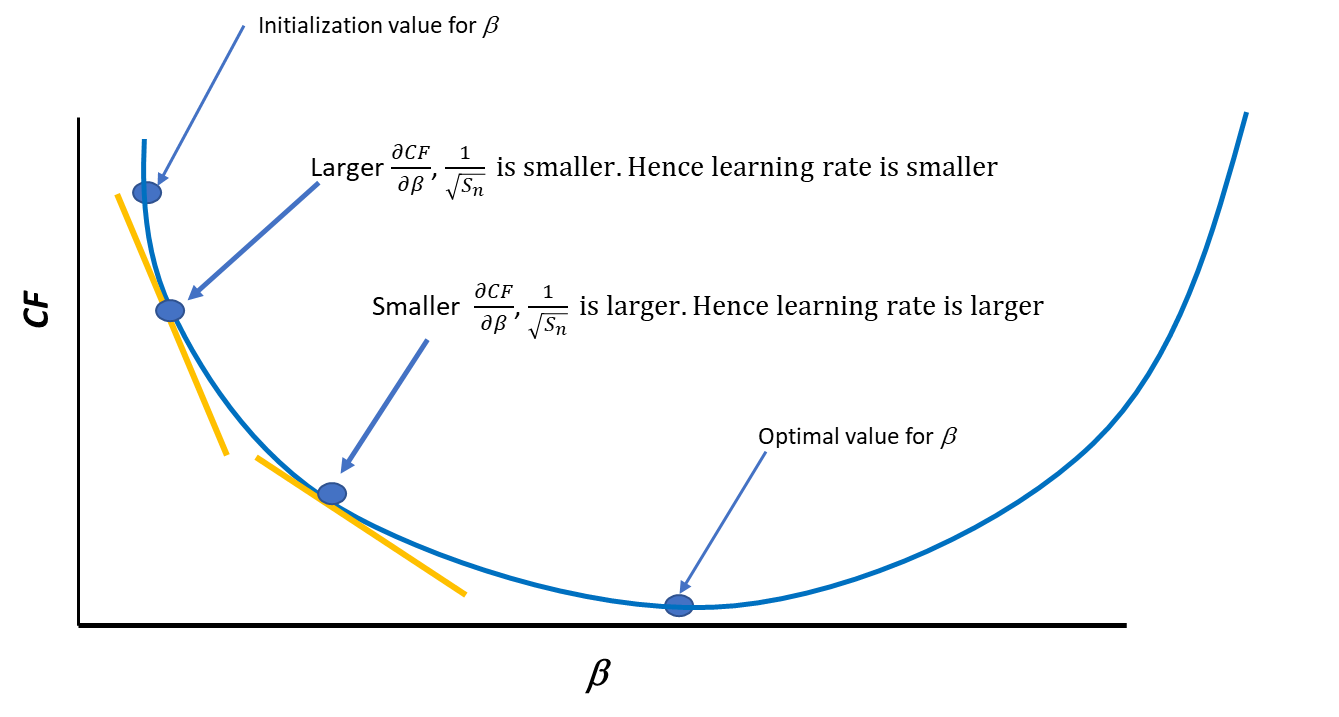
In this approach, the learning rate increases or decreases based on the gradient value of the cost function. For higher gradient value, the learning rate will be smaller and for lower gradient value, the learning rate will be larger. Hence, the learning decelerates and accelerates respectively at steeper and shallower parts of the cost function curve. The formula used in this approach is shown in the below equation.

learning rate

In the above equation, o is the initial learning rate and ‘sn’ is the momentum factor, which is calculated using below equation. ‘n’ is the epoch/iteration number

learning rate

In the above equation, 𝛾 is a hyperparameter whose value is typically between 0.7 and 0.9. Note that in equation-7, momentum factor Sn is an exponentially weighted average of gradients. So not only the value of the current gradient is considered, but also the values of gradients from the previous epochs are considered to calculate the momentum factor. Figure 5 shows the idea behind the gradient adapted learning rate. When the cost function curve is steep, the gradient is large, and the momentum factor ‘Sn’ is larger. Hence the learning rate is smaller. When the cost function curve is shallow, the gradient is small and the momentum factor ‘Sn’ is also small. The learning rate is larger.



The gradient adapted learning rate approach eliminates the limitation in the decay and the drop approaches by considering the gradient of the cost function to increase or decrease the learning rate. This approach is widely used in training deep neural nets with stochastic gradient descent.