REGEX
Machine Learning and Deep Learning Summer Internship
Assignment 6
ID: SIRSS2309
Name: K Manoj

Q1. Calculate/ derive the gradients used to update the parameters in cost function optimization for simple linear regression.

A cost function is something we want to minimize.

The coefficients used in simple linear regression can be found using stochastic gradient descent.

Linear regression is a linear system and the coefficients can be calculated analytically using linear algebra. Stochastic gradient descent is not used to calculate the coefficients for linear regression in practice (in most cases).

Linear regression does provide a useful exercise for learning stochastic gradient descent which is an important algorithm used for minimizing cost functions by machine learning algorithms.

As stated above, our linear regression model is defined as follows:

y = B0 + B1 * x

Gradient Descent Iteration #1

Let's start with values of 0.0 for both coefficients.

B0 = 0.0

We can calculate the error for a prediction as follows:

$$\text{error} = p(i) – y(i)$$

Where p(i) is the prediction for the i’th instance in our dataset and y(i) is the i’th output variable for the instance in the dataset.

We can now calculate he predicted value for y using our starting point coefficients for the first training instance:

$$x=1,y=1$$

$$p(i)=0.0+0.0*p(i)=0$$

Using the predicted output, we can calculate our error:

$$\text{Error}=0-1= -1$$

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

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:

$$B0(t+1) = B0(t) – \text{alpha} * \text{error}$$

Where B0(t+1) is the 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 and 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.

We can plug in our numbers into the equation and calculate the updated value for B1:

$$B1(t+1) = 0.0 – 0.01 * -1 * 1$$

$$B1(t+1) = 0.01$$

We have just finished the first iteration of gradient descent and we have updated our weights to be B0=0.01 and B1=0.01. This process must be repeated for the remaining 4 instances from our dataset.

One pass through the training dataset is called an epoch.

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

The cost function is a function of the parameters and when the sign is positive then the step will decrease:

$$W = W-(a \times dW)$$

when the sign is negative then the step will increase : New

$$b = b - (a \times -db)$$

$$b = b + (a \times db)$$

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

The mean squared error (MSE) tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them.

The squaring is necessary to remove any negative signs. It also gives more weight to larger differences.

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.
The drawback of MSE is that it is very sensitive to outliers. When high errors (which are caused by outliers in the target) are squared it becomes, even more, a larger error.

MSE can be used in situations where you don't need high errors.

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

Initial rate can be left as system default or can be selected using a range of techniques. A learning rate schedule changes the learning rate during learning and is most often changed between epochs/iterations. This is mainly done with two parameters: decay and momentum. There are many different learning rate schedules but the most common are time-based, stepbased and exponential.

Decay serves to settle the learning in a nice place and avoid oscillations, a situation that may arise when a too high constant learning rate makes the learning jump back and forth over a minimum, and is controlled by a hyperparameter.

Momentum is analogous to a ball rolling down a hill; we want the ball to settle at the lowest point of the hill (corresponding to the lowest error). Momentum both speeds up the learning (increasing the learning rate) when the error cost gradient is heading in the same direction for a long time and also avoids local minima by 'rolling over' small bumps. Momentum is controlled by a hyper parameter analogous to a ball's mass which must be chosen manually—too high and the ball will roll over minima which we wish to find, too low and it will not fulfil its purpose the formula for factoring in the momentum is more complex than for decay but is most often built in with deep learning libraries such as keras.

Time-based learning schedules alter the learning rate depending on the learning rate of the previous time iteration. Factoring in the decay the mathematical formula for the learning rate is:

$$nn+1 = nn / 1+dn$$

where n is the learning rate, d is a decay parameter and is the iteration step.

Step-based learning schedules changes the learning rate according to some pre defined steps. The decay application formula is here defined as:

$$nn = n0d\text{\^{}}floor(1+n/r)$$

Where nn is the learning rate at iteration n, n0 is the initial learning rate, d is how much the learning rate should change at each drop (0.5 corresponds to a halving) and r corresponds to the droprate, or how often the rate should be dropped (10 corresponds to a drop every 10 iterations). The floor function here drops the value of its input to 0 for all values smaller than 1.

Exponential learning schedules are similar to step-based but instead of steps a decreasing exponential function is used. The mathematical formula for factoring in the decay is:

nn=n0e^-dn

Where d is a decay parameter