Hackathon 1

Intro to Deep Learning CSCE 879

I have used a linear equation, $\boldsymbol{A * x + b = 0}$ **or** $\boldsymbol{A * x = -b}$ for all the examples that I have taken to examine on how learning rate and number of iterations affect the gradient descent optimization problem on updating the value of x and reducing the error. Values of A and b are fixed, and x is learnable variable that is continuously updated. The main goal that we are trying to solve and understand using this exercise is to change the values of hyperparameters such as learning rate, number of iterations in our program and see on how well x variable is updated by applying gradients in the Gradient descent optimization problem. I have taken a fixed value of A*, b* and continuously update the value of x* in the program using the code provided and check how well these hyperparameters perform. Considering No. of Iterations at first, I have taken a fixed learning rate value for the 1st-4 examples in my code (.py file) and observed on how well this hyperparameter performs on updating the value of x and reducing the overall sq. error throughout. I took a constant learning rate value of 0.15 and values 25, 100, 250 and 500 as the number of iterations in my 1st, 2nd, 3rd, 4th example. I observe that it is an important factor as it helps in reducing the Squared error gradually. Results show that sq error reduces with an increase in number of iterations which are performed to update the value of x variable by applying gradients using the Adam optimizer that is being used in the problem. Using the values above sq error was reduced from 10^0, 10^{-2}, 10^{-6}, 10^{-7} in Example 1-4. So, from the results obtained we get an intuition that sq. error is inversely proportional to num_iterations and if the number of iterations increase it leads to decrease in the sq. error. It can be said that, If the number of iterations is large enough, we will eventually learn a vector for x which approximately satisfies the system of equations. Considering the learning rate parameter, I have taken a constant value of num_iterations as 50 and taken multiple learning rate values as 0.05, 0.1, 1.0, 10.0 in the following 5th, 6th, 7th and 8th example in the code. It is noticed for these values of hyperparameters and using them for gradient descent algorithm to continuously update value of variable x and reducing the square error. For a very small learning rate value of 0.05 we observe that we get a low error value of 1.234 but it is greater than 0.398 that we have obtained with a learning rate value of 0.1. We observe values of 0.597 and 1.289 with learning rate values of 1.0 and 10.0. So, we observe from the results obtained in the later examples (i.e., 5th - 8th) that extreme small ($10^{-2}\ scale$) and larger values (10^1 scale) of learning rate do not perform as well as the learning rate values in the (10^{-1} scale) and (10^0 scale). The best results were obtained with a learning rate value of scale (10^{-1} scale) as observed when the program was run in most of the cases. These values give us an intuition that gradient descent optimization could be well performed with a low value i.e., between (10^{-1} scale) and (10^0 scale) of learning rate rather than extreme low ($10^{-2}\ scale$) or large ($10^1\ scale$) to update the value of variable x and solve a problem of linear equation.

Learning rate	Num_iterations	Sq. error
0.15	25	10^{0}
0.15	100	10^{-2}
0.15	250	10^{-6}
0.15	500	10^{-7}
0.05	50	1.234
0.1	50	0.398
1.0	50	0.597
10.0	50	1.289

- The results in the table represent values that are similar to the values obtained most of the time program was run
- Imp. Values in the table will always change as we are inserting random values in A, x and b every time they are defined in our code.

Also, the results obtained after each time the program was run were different as we are taking random values in the values that we use to perform the gradient descent optimization problem. The results stated in table are obtained for an instance when the program was run, and you will not get same results if you run the code again, but the basic intuition explained and to why these kind results were obtained is explained in above paragraph.

This code was used to generate random values for the values used in program

A* = tf.random.uniform([3,4], minval=-3, maxval=3)

x* = tf.variable(tf.random.normal([4, 1]))

b* = tf.random.uniform([3,1], minval=-3, maxval=3)

Puranjit Singh
94849312