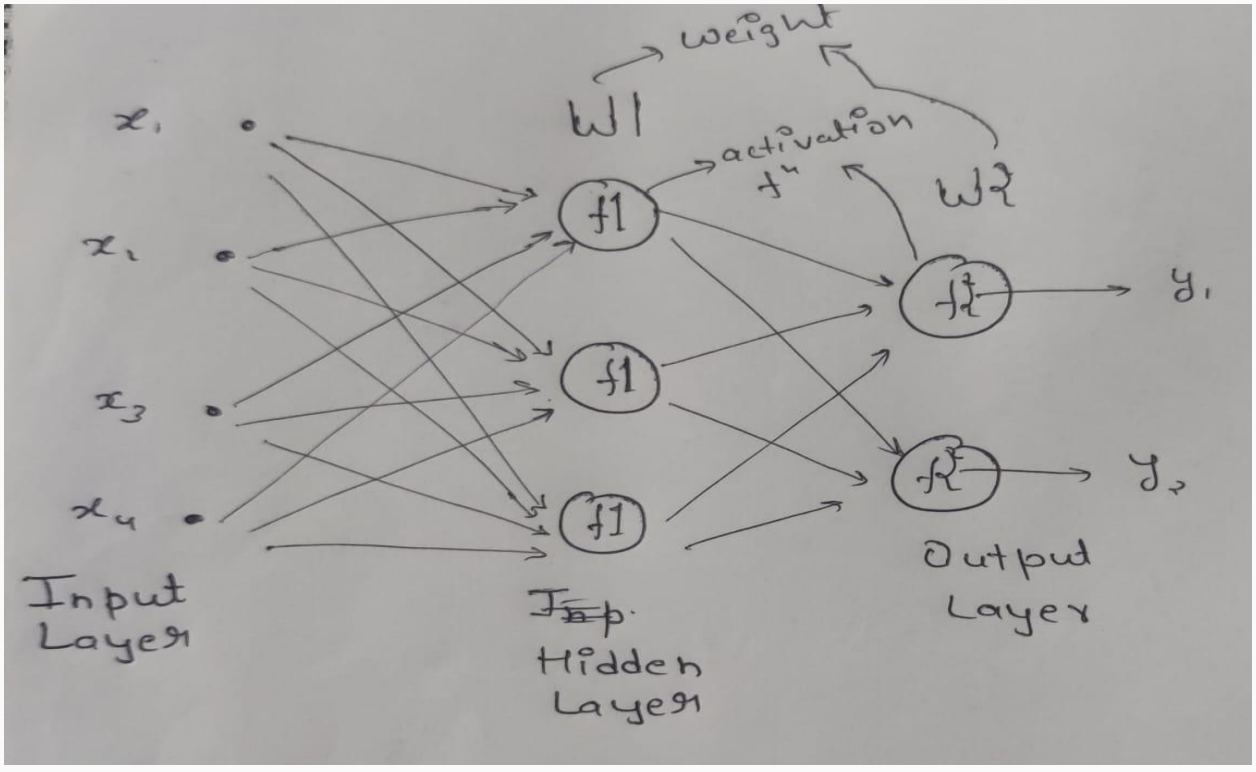
Task II Question 2

Problem Statement:

Create a simple class to build the multi-layer perceptron network for the function y = x*x.
the constructor will have one list as an input example(arr=[100,500,600,7])
arr[0] is the input size of the neural network.
arr[1:-1] number of neurons in each hidden layer
a[-1] number of neurons of the output layer

Detail Approach:

Creating a Neural Network:
A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.



Input Layer in Neural Network : Initial Data for Neural Network

Hidden Layers in Neural Network : A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function.

Hidden Layers in Neural Network : The output layer takes in the inputs which are passed in from the layers before it, performs the calculations via its neurons and then the output is computed.

Weights In the Neural Network : The parameters of a neural network are typically the weights of the connections. In this case, these parameters are learned during the training stage. So, the algorithm itself (and the input data) tunes these parameters.

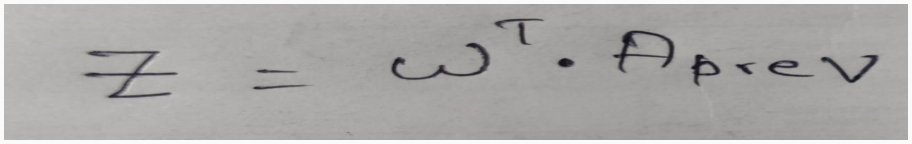
Activation Function in the Neural Network : An Activation Function decides whether a neuron should be activated or not.

Loss Function and Cost Function : A loss function/error function is for a single training example/input. A cost function, on the other hand, is the average loss over the entire training dataset.

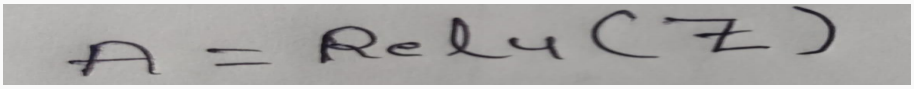
Learning Rate : the learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function

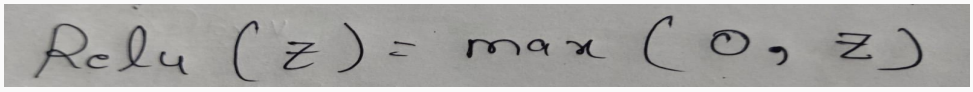
Neural Network Working : Neural network is a set of neurons organized in layers. Each neuron is a mathematical operation that takes it's input, multiplies it by it's weights and then passes the sum through the activation function to the other neurons. Neural network is learning how to classify an input through adjusting it's weights based on previous examples. These are some steps to show the working of Neural Network.

1. **Forward Propagation :** Forward propagation refers to the calculation and storage of intermediate variables (including outputs) for the neural network in order from the input layer to the output layer. First we multiply Input from the previous layer with Weights shown here.

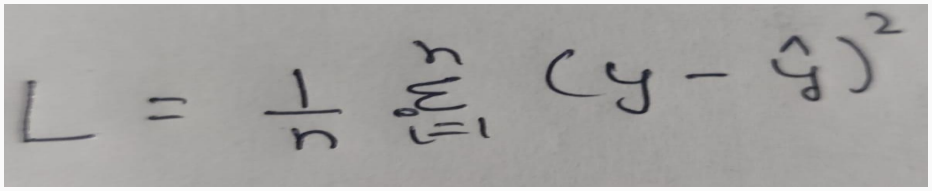


After that we apply the Activation Function to the above output. I am using Relu Function in this task.





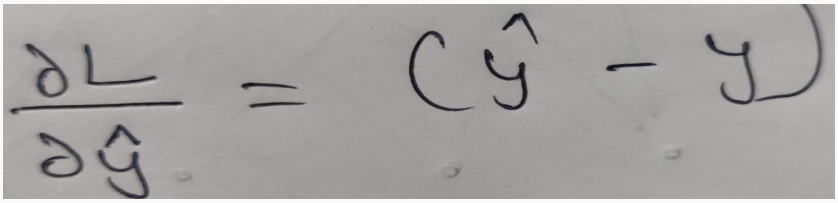
2. Calculation of the Cost : For this I used the MSE(mean square error) which can we calculated as :



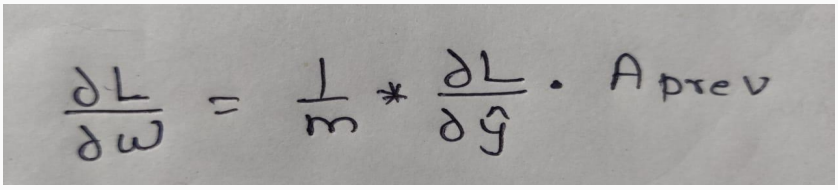
3. Back Propagation : Backpropagation refers to the method of calculating the gradient of neural network parameters. In short, the method traverses the network in reverse order, from the output to the input layer, according to the *chain rule* from calculus. The algorithm stores any intermediate variables (partial derivatives) required while calculating the gradient with respect to some parameters.

Calculation of Gradients :

- **Gradients with respect to output :**

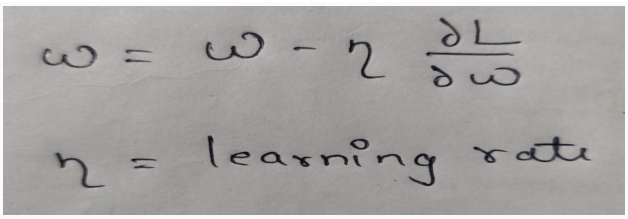


- **Gradient with respect to weights :**



After calculating these gradient I just update the weight which can we updated as :

Update the Weights :



Training of the neural network for given task :

1. **Batch SGD Backpropagation:**

   For each epoch:

   Loss = 0

   Loss_list = [ ]

   For batch in training_data:

   - **Take these batch as input**
   - **Pass these batch in forward network**
   - **Calculate cost**
   - **Calculate all the gradients**
   - **Update weights**
   - **Increase loss such that Loss = Loss+cost**

   Lost_list.append(loss)

2. **Single Example Backprop**

For each epoch:

Loss = 0

Loss_list = [ ]

For Single_example in training_data:

- **Take Single_example as input**
- **Pass this Single_example in forward network**
- **Calculate cost**
- **Calculate all the gradients**
- **Update weights**
- **Increase loss such that Loss = Loss+cost**

Lost_list.append(loss)

So that's how I train my deep learning model. Now take a look into the results and visualization.

Results

For testing I created dataset such that:

X.shape = (8,4)

Y.shape = (8,4)

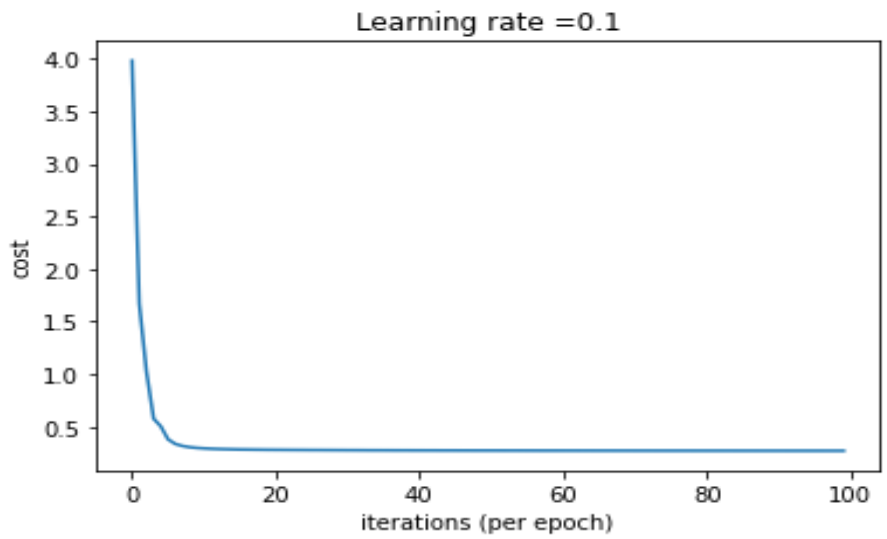
And I take :

Hidden_layer_neurons = [500, 600]

After training for 100 epochs I got plot between loss and epochs :

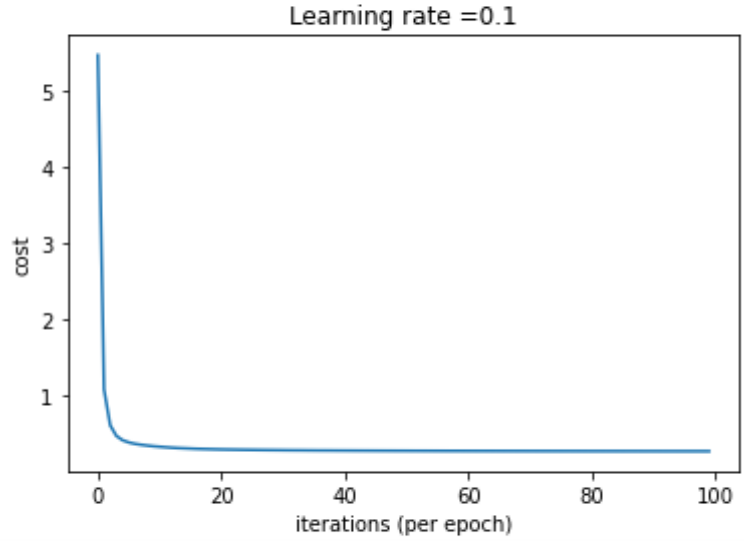
1. **Batch SGD Backpropagation**

```
loss after 98 : 0.2717866974335695
loss after 99 : 0.2717604994411344
```



2. Single Example Backprop

```
loss after 98 : 0.2739620226006755
loss after 99 : 0.2738929319947261
```



Future Tasks to Improve Performance:

- During my task I found the performance of the network mainly depends on how you initialize the weights. So Give task to use different techniques to initialize the weights.
- Give the task to implement deep learning optimizers.
- Give the task to use hyperparameters tuning to use different value of “learning rate” to get best performance.