```
# Authors: Amit Jadhav and Samyak Sheth
# Part of Advance Concepts in Machine Learning Course
# Edited: 05th November 2022
```

Defining and Initializing neural network weights and biases

In this assignment a 3 layered neural network is created. It contains of -

- One input layer with 8 nodes
- One hidden layer with 3 nodes
- One output layer with 8 nodes

This neural network is trained by epochs of forward and backward propogations where the associated weights and biases are updated according to the cost function.

The **reset_network function** below initialises the weights and biases to random values so that they can be trained. It creates matrices of the desired dimensions and assigns random values to them.

Forward-Propagation

For training the neural network, in the a forward propogation it uses an activation function, $\sigma(z)$ that calculates each activation of the network using feed-forward based on the following feed-forward equations

$$\mathbf{z}^{(n)} = \mathbf{W}^{(n)}\mathbf{a}^{(n-1)} + \mathbf{b}^{(n)}$$

$$\mathbf{a}^{(n)} = \sigma(\mathbf{z}^{(n)})$$

The **logistic sigmoid function** is used as the activation function.

$$\sigma(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$$

Back-Propagation

The 4 functions in the following cell constitutes the Back Propogation for this neural network. It starts from the output layer and updates the wieghts and bias here. The following derivations are formulated by the chain rule and is used in the update.

For weights:

$$\frac{\partial C}{\partial \mathbf{W}^{(21)}} = \frac{\partial C}{\partial \mathbf{a}^{(2)}}\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(12)}}\frac{\partial \mathbf{z}^{(12)}}{\partial \mathbf{W}^{(21)}}$$

For bias:

$$\frac{\partial C}{\partial \mathbf{b}^{(2)}} = \frac{\partial C}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(12)}} \frac{\partial \mathbf{z}^{(12)}}{\partial \mathbf{b}^{(2)}}$$

With the partial derivatives taking the form,

$$\frac{\partial C}{\partial \mathbf{a}^{(2)}} = 2(\mathbf{a}^{(2)} - \mathbf{y})$$

$$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(12)}} = \sigma'(z^{(12)})$$

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(21)}} = \mathbf{a}^{(1)}$$

$$\frac{\partial \mathbf{z}^{(12)}}{\partial \mathbf{b}^{(2)}} = 1$$

For the next layer however, the derivals look like -

$$\frac{\partial C}{\partial \mathbf{W}^{(10)}} = \frac{\partial C}{\partial \mathbf{a}^{(2)}} \left(\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}} \right) \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(01)}} \frac{\partial \mathbf{z}^{(01)}}{\partial \mathbf{W}^{(10)}}$$

$$\frac{\partial C}{\partial \mathbf{b}^{(1)}} = \frac{\partial C}{\partial \mathbf{a}^{(2)}} \left(\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}} \right) \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(01)}} \frac{\partial \mathbf{z}^{(01)}}{\partial \mathbf{b}^{(1)}}$$

With this we have moved one layer back in the network and here there is an extra term in the parantheses which takes the form:

$$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}} = \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(12)}} \frac{\partial \mathbf{z}^{(12)}}{\partial \mathbf{a}^{(1)}} = \sigma'(\mathbf{z}^{(12)}) \mathbf{W}^{(21)}$$

Training the Network

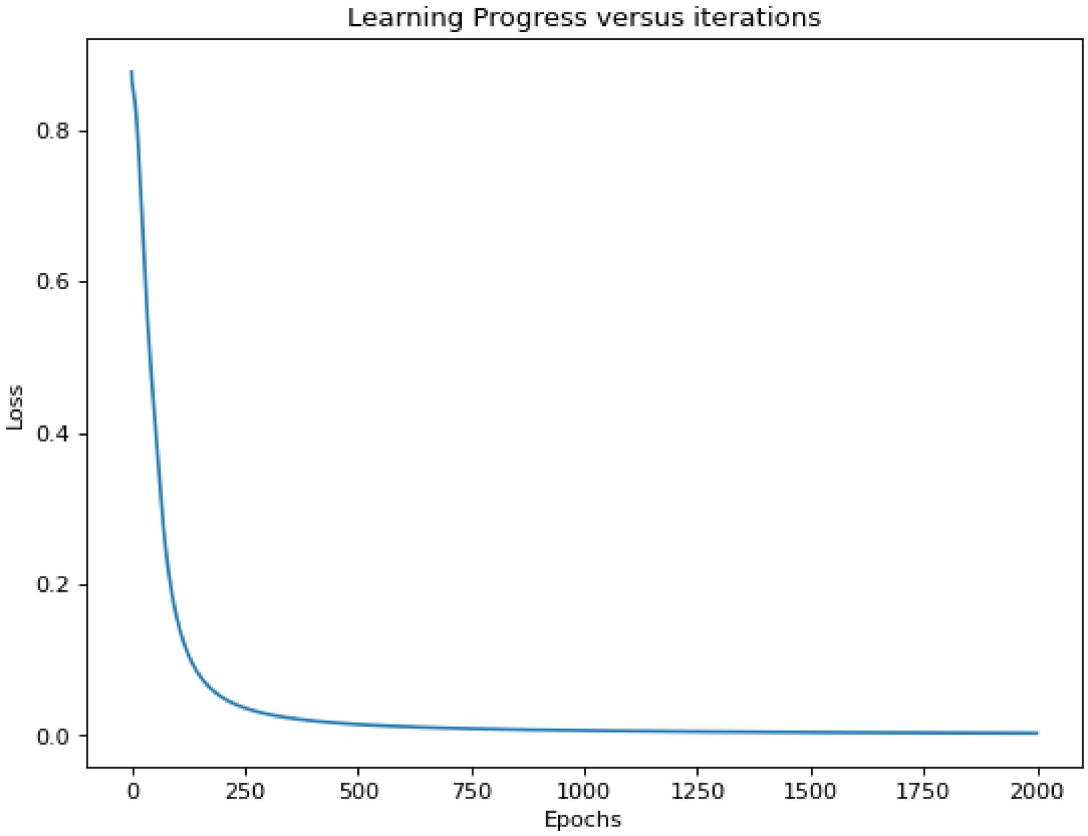
The training of the network takes place here. This is the initial training block where the learning rate is set to 1 and the iterations are set to 2000.

These parameters are tweaked in the experiments done below.

From the make_input_data() function the data is created and this is passed through the network by forward and backward propogations for a certain number of iterations where the weights and biases keeps getting updated after every back propogation step for each input-output pair. When all the iterations are done, the final learned weights and biases are printed.

PREDICTION ERROR ACCROSS EPOCHS (ITERATIONS)

```
from matplotlib.pyplot import figure
figure(figsize=(8, 6), dpi=80)
plt.plot(loss)
plt.title("Learning Progress versus iterations")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



EXPERIMENTS - Tweaking the learning rate or the number of iterations

EXPERIMENT 1

This experiment trains the network with a lower learning rate(0.5) keeping the iterations the same (2000) as the initial training method

EXPERIMENT 2

This example trains the network with a large number of iterations (4000) keeping the learning rate (1) same as the initial training method

EXPERIMENT 3

This experiment trains the network with changing both the learning rate and the number of iterations, i.e. More iterations (4000) with a lower learning rate (0.5)

EXPERIMENT 4

The back propogation here is modified in such a way that while updating the weights while mulitplying the derivatives with the learning weight there is an additional 10% noise (of the magnitude of the gradient) which is included into the gradient keeping the learning rate and number of iterations same as the initial training method, i.e. 1 and 2000 respectively.

INTERPRETATION OF CONCEPTS & EXPERIMENTAL RESULTS:

Neural network as a generalization of Logistic regression:

A single neuron is a (w,b, z, Sigmoid, a) structure where 'w' is the weight of the connection between previous and current layer, 'b' is the bias added to the net weighted sum of activations of previous layer neurons, 'z' is the total including weight sums and bias, 'sigmoid' is an example function to add non-linearity and restricting the output to a limited range, and 'a' is the final activation value reached at current neuron after applying the 'sigmoid' or any other activation function to 'z'. It also makes sense to say that this neuron resembles one logistic regression model that predicts a probabilistic value between two classes given training samples. This line of thought can thus be extended to generalize that a neural network can be thought of as a network of multiple logistic regression models.

Diminishing magnitude of prediction error while backpropagating the error in a neural network:

The error between actual and predicted output when backpropagated reduces in magnitude which is due to the sigmoid activation function being restricted from to a decimal value between 0 to 1. The chain rule of derivative multiplication to propagate the error gradient back through the network means more multiplications with more layers and hence a smaller value closer to 0.

Implication of learning rate (alpha):

Normally the learning rate maybe very low (0.0001 or lower) in a cost landscape with multiple local optima, for instance like a rastrigin function. In our example however, its a perfect world and all the training examples are available for training, thereby prompting for a very aggressive(high) learning rate. We experimented and observed that given same number of iterations, a higher learning rate gave a lower error in prediction.

Experimental results with number of iterations:

It was experimentally confirmed that higher the number of iterations, lower the error but higher the computational power consumption. So a threshold can be set on the error to stop the convergence.

Adding noise to the gradient (first derivative) of weights and biases:

To model a noise factor in the gradient measurement, 10% of the magnitude of the gradient was added / substracted from it. It was observed that this ensures quicker convergence.