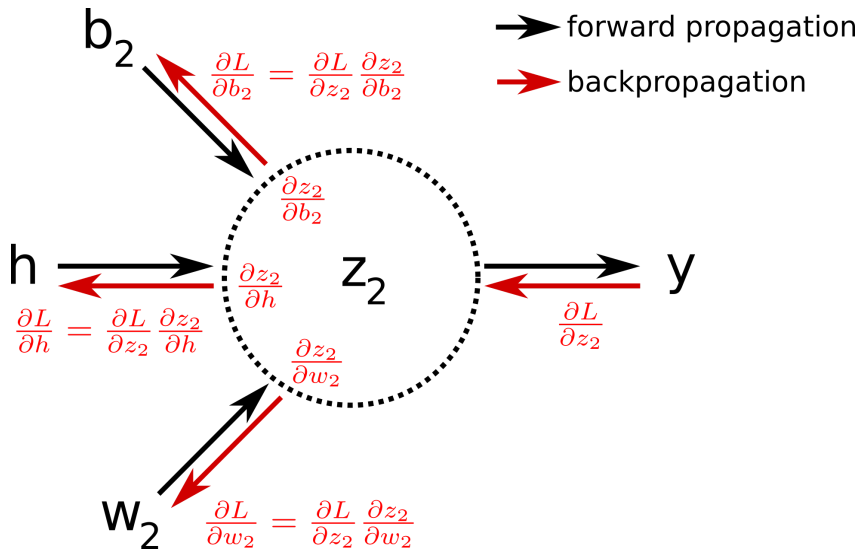
CS F464 | Machine Learning
Aditya Agarwal (2017B1A71075H)
Manisha Kataria (2017B3A70354H)
Jalaj Bansal (2017B3A71610H)
Assignment 2

B. Artificial Neural Network

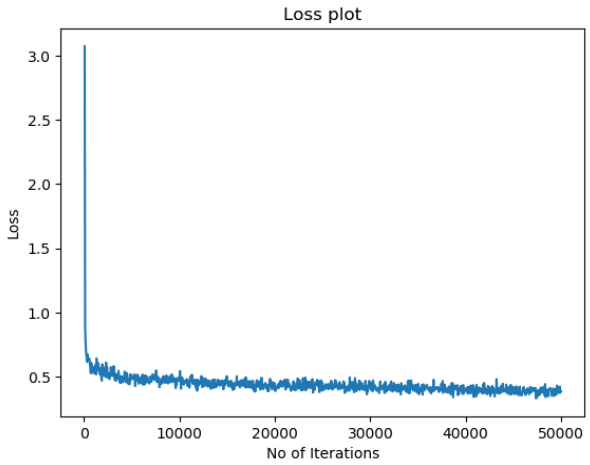
1. A very brief description of your model and its implementation.
   - Libraries used - pandas, numpy, math, matplotlib.
   - We have built a classifier using an artificial neural network architecture for the multi class dataset to perform classification for new samples.
   - We first normalize the features since there is quite a difference in scales between the feature values.
   - We create one hot labels for the samples and separate the labels from the feature set.
   - We randomly split the dataset into train and test data in the ratio 70:30.
   - The buildModel() function takes the following parameters as input to avoid hardcoding the model hyperparameters -
     - Learning rate
     - Number of hidden layers
     - Number of nodes in hidden layers
     - Number of epochs
   - One Hidden layer - Initialize random weights for hidden layer and output layer.
   - Two Hidden layers - Initialize random weights for the hidden layers and output layer.
   - Feedforward phase - Multiply inputs with weights and feed to the next layer. Next layer applies activation function to get outputs. We used sigmoid activation function for hidden layers and softmax function for output layer.
   - Backpropagation phase - Calculate deltas for each layer by multiplying the error at the layer with the previous weights/inputs.

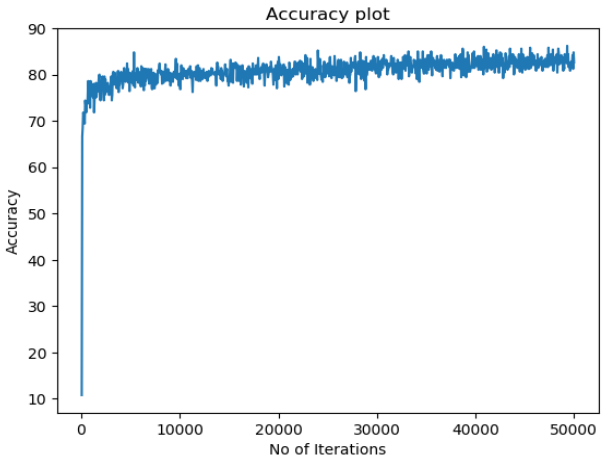


- Updating weights and biases -
    - W = W - ETA* (sigma(inputs).T @ delta)
    - B = B - ETA*sum(delta)
- After every 50 epochs, calculate and print -
    - Loss = sum(actual_outputs * log(predicted outputs))/number of samples
    - Accuracy = Correctly classified/Total samples
- For mini batch gradient descent, we select x random samples from the dataset at every epoch where x = batch size. Then train on this batch.

2. Hyperparameters
    - Sigmoid activation function for hidden layers and softmax function for output layer.
    - Users are allowed to specify hyperparameters.
    - We obtained varied results for different learning rates, hidden nodes, hidden layers.
    - As we increase the number of hidden nodes from 16 to 256 in the hidden layers, training accuracy increases linearly and reaches upto 100%. But it must be noted that the testing accuracy decreased for a large number of hidden nodes.
    - For a slower learning rate (<0.0001), accuracy increases very slowly and requires a large number of iterations for good results. For a high learning rate (0.1), accuracy is irregular as after each iteration we may move towards or away from the local minima. An optimal learning rate is around 0.001 which leads to a testing accuracy upwards of 76%.
    - We originally obtained better results with one hidden layer compared to two hidden layers but after varying hyperparameters, we were able to obtain similar results.

3. Train and Test Metrics
    - One hidden layer
        - Training accuracy = 81.2% (average)
        - Test accuracy = 76.1% (max)
        - Loss - 0.5561 (min)
    - Two hidden layers
        - Training accuracy >95% (with >64 nodes)
        - Test accuracy - [70-75]% (range with different hyperparameters)
        - Loss - 0.4595 (min)





4. Accuracy plots for three different learning rates and 1. 1 hidden layer, 128 hidden nodes and batch size = 50. 2. 2 hidden layers, 64 hidden nodes and batch size = 100.