Artificial Neural Network

-Rahul Mehndiratta 2017A7PS1479H
-BVS Ruthvik2019A7PS0017H
-R Vedang.   2019A7PS0150H

Problem Statement:

1. In this assignment, you will have to a implement a simple artificial neural network with at most two hidden layers from scratch that can perform multi-class (1-out-of-K) classification. Choose appropriate non-linear activation (such as sigmoid, tanh, ReLU, LeakyReLU, etc) functions for the hidden and output layers and try to achieve as good accuracy as possible.
2. Use Stochastic Gradient Descent or Mini-Batch Gradient Descent (with appropriate batch size) to train the model. As usual, you can make a random 70:30 split on the given data and use it for training and testing respectively. Plot the loss and accuracy of your model (advisedly in two separate plots, i.e. one for accuracy and one for loss) after every 50 iterations to visualize the training better.
3. Try to vectorize your code as much as possible to make your computations faster and efficient. Do not hard code any parts of the implementation unless it is absolutely necessary.

Introduction :

Model takes the dataset and splits it into 70:30 ratio for training-testing, respectively. In our dataset, we had 6 independent variables(attributes) and one dependent variable(class). Min-max normalization is used here for feature scaling. Weight vectors and bias vectors take random value in the beginning. First of all, dot product of input vector(xi) and hidden layer weight vector(wh) is calculated (along with bias value(bh)). The result is then passed on to an activation function ReLU. We then take the dot product of the output of the activation function and the weights of the output layer(wo) and add the bias value(bo). The result is once again passed on to an activation func at the output layer. Softmax function is chosen here. Once we get the output, we compare it with the original/given value of the output and calculate the cost function using

Cross Entropy Loss Function and then try to minimize the cost function by changing the weight vectors accordingly with an appropriate learning rate.

Similar process is repeated for the second hidden layer in the 2 hidden layers ANN. The activation func used for the second hidden layer is again ReLU.

Hyper-parameters for the 1 Hidden Layer ANN model:

Number of hidden layers = 1
Number of units per layer = 14 for the hidden layer and 10 for the output layer
Activation functions for the hidden layer = ReLU for hidden layer and Softmax for output layer
Learning rate = 0.0001
Using Batch Gradient descent
Batch Size=Size Of Training Set=1400

Hyper-parameters for the 2 Hidden Layers ANN model:

Number of hidden layers = 2
Number of units per layer = 13 for 1st and 8 for 2nd hidden layer and 10 for the output layer
Activation functions for the hidden layer = ReLU for both the hidden layers and Softmax for output layer
Learning rate = 0.00003
Using Batch Gradient descent
Batch Size=Size Of Training Set=1400

Observations:

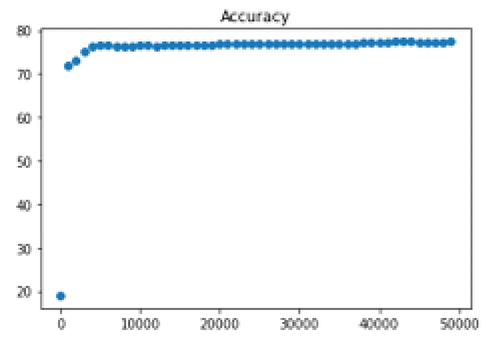
The final train metrics for 1 Hidden Layer ANN model: **Loss = 0.524 and Accuracy = 78.57142857142857**

The final test metrics for 1 Hidden Layer ANN model: **Accuracy = 70.50%**

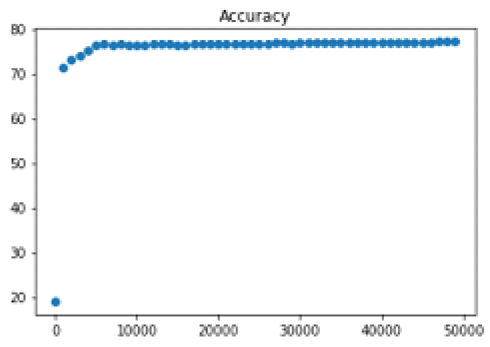
The final train metrics for 2 Hidden Layers ANN model: **Loss = 0.537 and Accuracy = 78.0%**

The final test metrics for 2 Hidden Layers ANN model: **Accuracy = 71.667%**

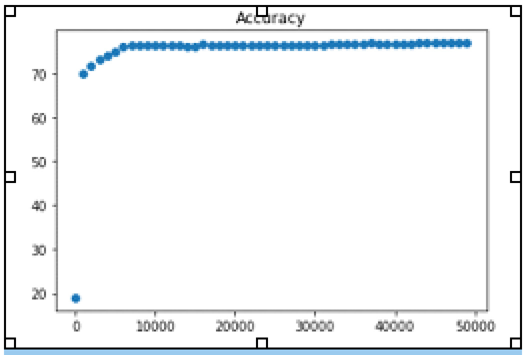
Plots of accuracy for 1 Hidden Layer ANN model (with 3 different learning rates):



Learning Rate = 0.0001

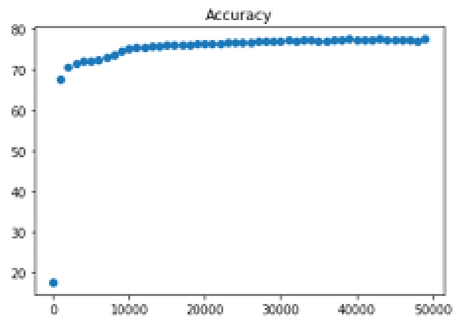


Learning Rate = 0.00008

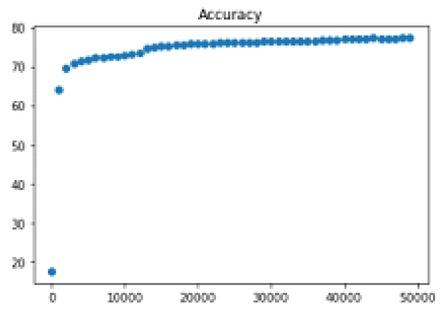


Learning Rate = 0.00006

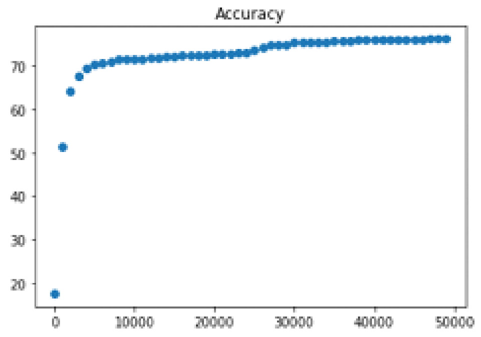
Plots of accuracy for 2 Hidden Layer ANN model (with 3 different learning rates



Learning rate = 0.00003



Learning rate = 0.00002



Learning Rate = 0.00001