Assignment 2 Report

Harsh Vora
CWID : A20445400
CS577 – S20

March 2, 2020

Problem Statement: To implement a neural network in python by not using a GPU framework.

Implementation details:

Import all packages that are necessary such as numpy and pandas.

Addition Class :

Forward Pass:

Forward pass implements Matrix Addition.

Backward Pass:

Backward Pass gets the incoming gradient which is propagated backwards.

Multiply Class :

Forward Pass:

Forward Pass implements Matrix Multiplication.

Backward Pass:

Backward Pass gets the adjoining vector and incoming gradient or the Kronecker product of the adjoining vector and incoming gradient which is propagated backwards.

Sigmoid Class:

Forward Pass:

Forward pass implements Sigmoid function.

Backward Pass:

Backward Pass calculates derivative of sigmoid function.

Softmax Class:

Forward Pass:

Forward pass implements Softmax function.

Backward Pass:

Backward Pass calculates derivative of Softmax function.

Categorical Cross Entropy Class:

Forward Pass:

Forward Pass calculates categorical cross entropy loss.

Backward Pass:

Backward Pass calculates derivative of categorical cross entropy loss.

L2 Class:

Forward Pass:

Forward pass implements L2 Loss.

Backward Pass:

Backward Pass calculates derivative of L2 loss.

Computational Graph Class It implements the Computation Graph

Init:

This method initializes the weights of the graph and the number of nodes for the hidden and output layers. It also defines all the classes which will be used in the class.

Forward:

Forward method obtains the loss by performing the forward pass on the computation graph.

Backward:

Backward method obtains the gradients for each layer by performing the backward pass on the computation graph.

Train Network:

The train_network method gathers the input and train the network for fixed number of epochs it also performs Stochastic Gradient Descent(SGD).

Test Network:

The test_network method shows the accuracy for the test data by testing the network.

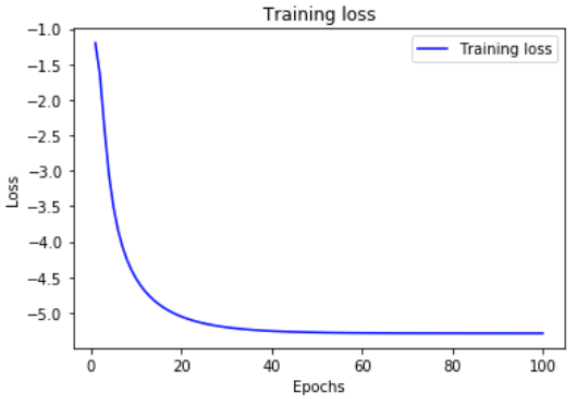
Test Cases:

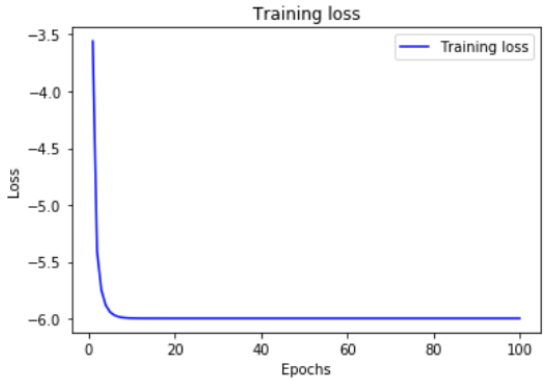
The network consists of 3 layers. One input layer, one hidden layer and one output layer. The input layer and hidden layer uses sigmoid activation. The output layer uses softmax activation. All the layers use 3 nodes.

Below are my results for Balance-Scale dataset:

https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data

Learning Rate	Decay Rate	Input Layer Nodes	Epochs	Accuracy
0.1	0.01	4	100	0.53
0.01	0.001	4	100	0.52

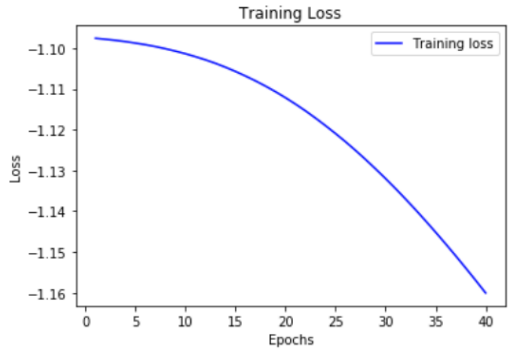


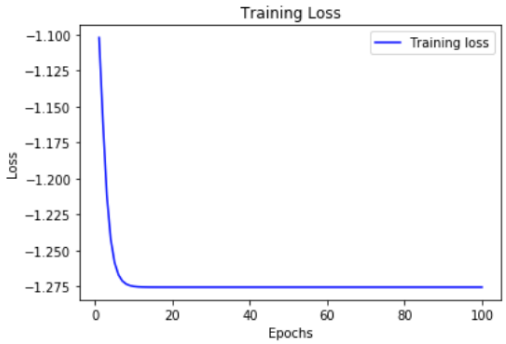


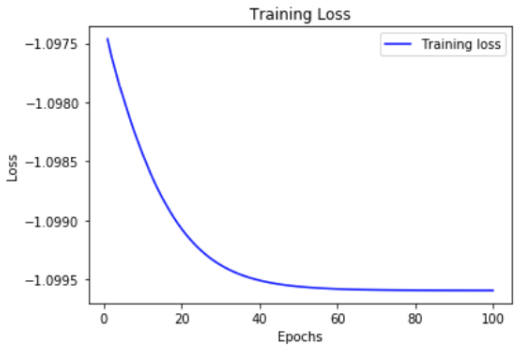
Below are my results for Iris Dataset.

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

Learning Rate	Decay Rate	Input Layer Nodes	Epochs	Accuracy
0.1	0.01	4	100	0.36
0.01	0.001	4	100	0.62
0.01	0.001	4	40	0.68







Conclusion:

According to my observation as I increase the decay rate and decrease the epoch size my model does not overfit and the model is able to train with higher accuracy.