BITS F464 : Machine Learning
Assignment - 1
Artificial Neural Networks

Omkar Pitale	Aditya Chopra	Anuradha Pandey
2019A7PS0083H	2019A7PS0178H	2019A7PS0265H

April 2021

1 Introduction

Neural network methods share a loose inspiration from biology in that they are repre- sented as networks of simple neuron-like processors. A typical neuron takes in a set of inputs, sums them together, takes some function of them, and passes the output through a weighted connection to another neuron.
In this report, we try to analyse the performance of Neural network model implemented from scratch using NumPy, Pandas and Matplotlib on the given dataset.

2 Dataset and Data Preprocessing

Only one dataset was provided as described in **Dataset Table**. The dataset was sampled as PyData Dataframe, and then converted to two NumPy Arrays: Feature vectors and Targets. The Feature vectors and Targets were split into 70:30 ratio as asked in the question for training and testing dataset.

Data scaling techniques used -
1. Standardization : $x = \frac{x - \mu}{\sigma}$
2. Normalization : $x = \frac{x - \min(x)}{\max(x) - \min(x)}$

Name	dataset_NN.csv
Type	csv
Number of features	6
Total Classes	10
No. of examples in each class	200
Total sample size	2000

Dataset Table : Description of the Dataset

3 Network Architecture

3.1 Two Layer Neural Network

We built a neural network model with one hidden layer and experimented with different activations like Sigmoid, Tanh, ReLu, Leaky ReLU. The last layer had activation of Softmax for multi-class classification. The weights were initialized randomly as well as with 'He' initialization when ReLu activation was used. For training we used both Stochastic Gradient descent as well as Mini-Batch gradient descent with momentum. The loss was observed to be highly fluctuating beyond 20000 epochs while using Mini-Batch and above 10000 while using Stochastic gradient descent. The highest training accuracy achieved by this model was 81% with average of 74% and the loss for this configuration was observed to be 0.024 per data point .

3.2 Three Layer Neural Network

This model was built using two hidden layers and among the 100+ runs the three layered Neural network performed slightly better than the previous model. We used power of 2 method to set the number of hidden units in each layer. Training with more epochs yielded a better accuracy. The highest accuracy achieved by this model under various hyperparameters was 89% with an average of 74% and the loss for this configuration was observed to be 0.019 per data point.

3.3 Observations

The observations made while training the data with the models were -

1. In general, a bigger network performs better than a smaller network because when you increase the number of layers, you actually provide a more trainable parameters which helps the network fits the model in a complex way.

2. Using a bigger network required more iterations to train. Also there's a lot of chances that you might end up overfitting the data.

3. We used three different initializers for the weights i.e. Random, He, Xavier and all these initializations led to convergence of the model at different local minima.

4. A lower learning rate requires more iterations to converge, but conversely, higher learning rate might lead to fluctuation in the exploding gradient. For this we used clipping techniques for numerical stability.

5. Different activation functions for the hidden layers provides drastically varying results. We significantly used ReLu as it gave more trustable results. Sometimes the ReLu suffered from dying neuron probelm so in such a scenario to rectify the problem we used Leaky ReLu but it didn't change the accuracy of the model that much.

4 Equations used

4.1 Activation Functions

1. Sigmoid : $y = \dfrac{1}{1+e^{-x}}$
2. tanh : $y = \tanh(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
3. ReLu : $y = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$
4. Leaky ReLu : $y = \begin{cases} \alpha \cdot x & x \leq 0 \\ x & x > 0 \end{cases}$ with $\alpha = 0.01$ generally.
5. Softmax : $y = \dfrac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}}$ $\forall i \in \{0, 1, \cdots, k-1\}$ where k is the number of classes.

4.2 Loss Function

Cross Entropy Loss for multiple classes : $\mathcal{L}(\hat{y}, y) = -\sum_{j=0}^{k-1} y_j \log(\hat{y_j})$

Total Cost Function : $J = \dfrac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}, y) = -\dfrac{1}{m}\sum_{i=1}^{m}\sum_{j=0}^{k-1} y_j \log(\hat{y_j})$

4.3 Gradient Descent

1. Batch Gradient Descent : $\begin{cases} w^{\tau+1} = w^{\tau} - \eta \cdot \nabla_{w^{\tau}} J(w, b) \\ b^{\tau+1} = b^{\tau} - \eta \cdot \nabla_{b^{\tau}} J(w, b) \end{cases}$
2. Stochastic Gradient Descent : $\begin{cases} w^{\tau+1} = w^{\tau} - \eta \cdot \nabla_{w^{\tau}} J(w, b; x^{(i)}, y^{(i)}) \\ b^{\tau+1} = b^{\tau} - \eta \cdot \nabla_{b^{\tau}} J(w, b; x^{(i)}, y^{(i)}) \end{cases}$
3. Mini Batch Gradient Descent : $\begin{cases} w^{\tau+1} = w^{\tau} - \eta \cdot \nabla_{w^{\tau}} J(w, b; x^{(i:i+n)}, y^{(i:i+n)}) \\ b^{\tau+1} = b^{\tau} - \eta \cdot \nabla_{b^{\tau}} J(w, b; x^{(i:i+n)}, y^{(i:i+n)}) \end{cases}$

4.4 Optimizers

Momentum : $\begin{cases} M_{\tau+1} = \beta M_{\tau} + (1-\beta)\nabla J \\ w^{\tau+1} = w^{\tau} - \eta M_{\tau+1} \\ b^{\tau+1} = b^{\tau} - \eta M_{\tau+1} \end{cases}$

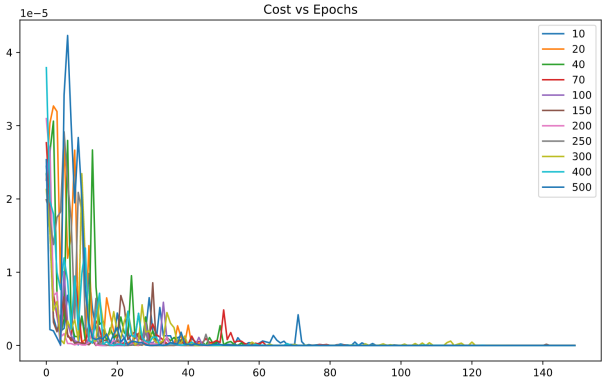
5 Results

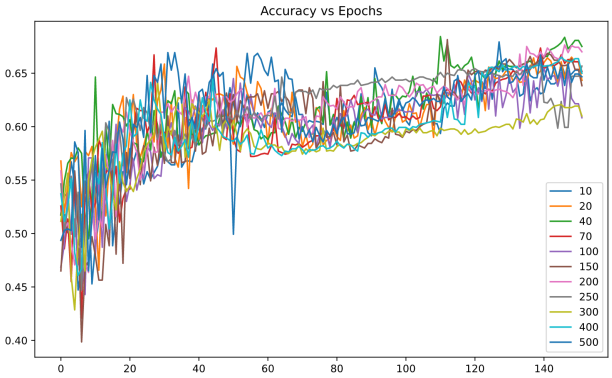
5.1 Two Layer Neural Network

Tested under different epochs and learning rates with Sigmoid, tanh, ReLu, Leaky ReLu activations.

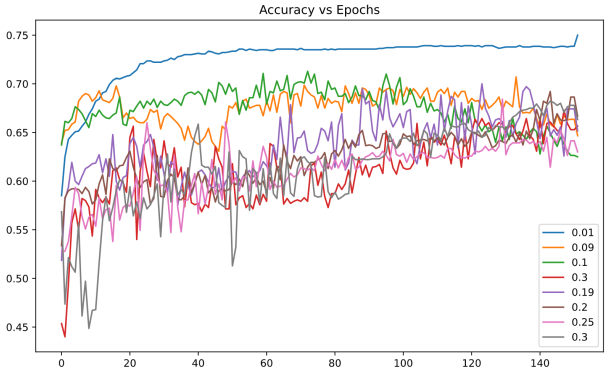
5.1.1 Stochastic Gradient Descent

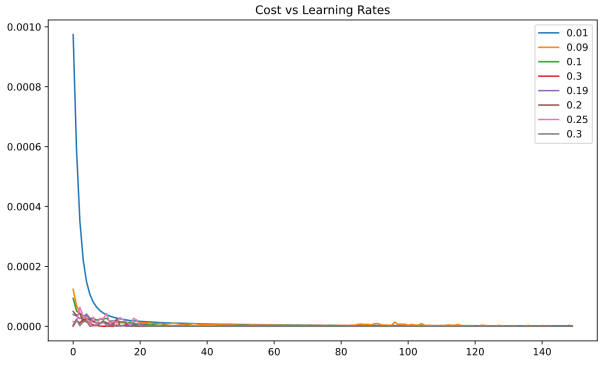
- With different epochs





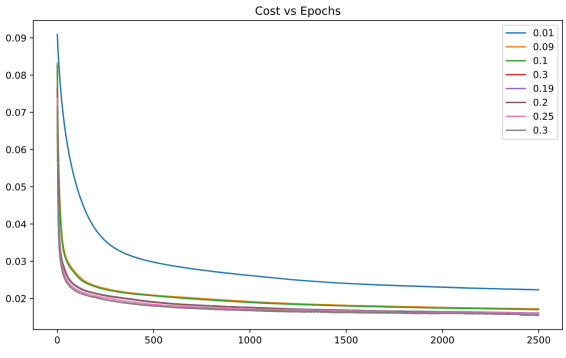
- With different learning rates

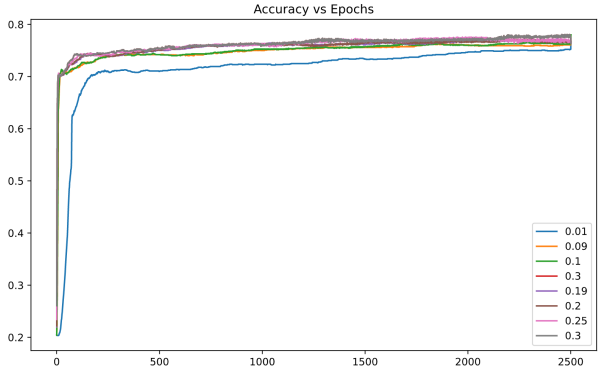




5.1.2 Mini-Batch Gradient Descent

- With different learning rates



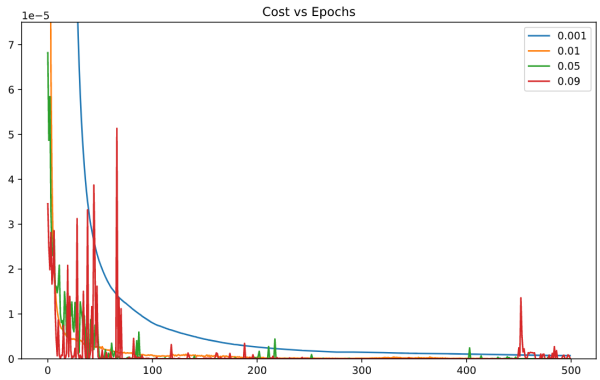


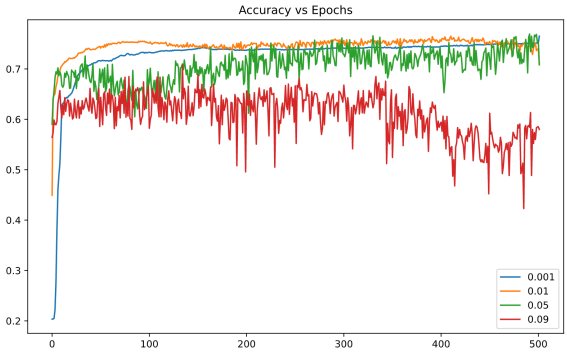
5.2 Three Layer Neural Network

Tested under different epochs and learning rates with Sigmoid, tanh, ReLu, Leaky ReLu activations.

5.2.1 Stochastic Gradient Descent

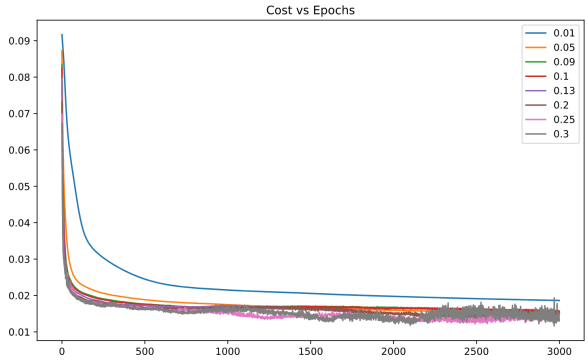
- With different learning rates

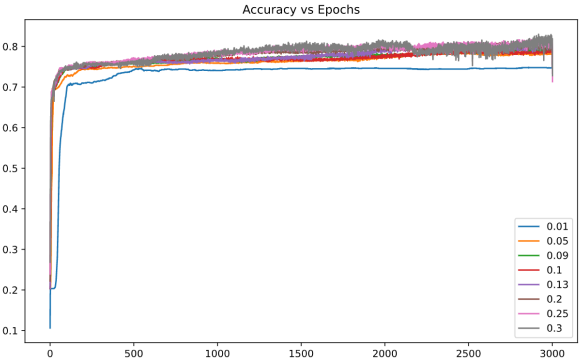




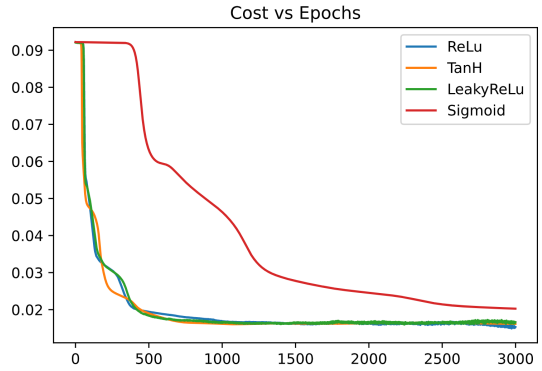
5.2.2 Mini-Batch Gradient Descent

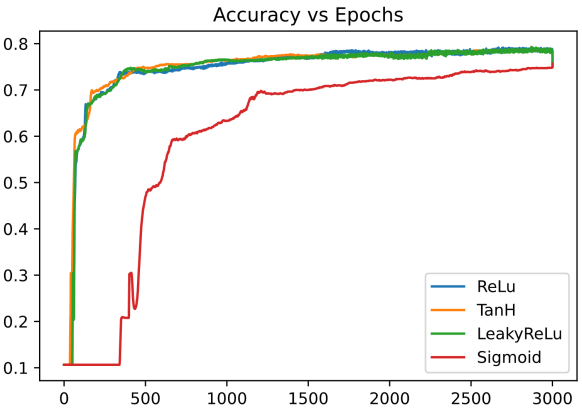
- With different learning rates



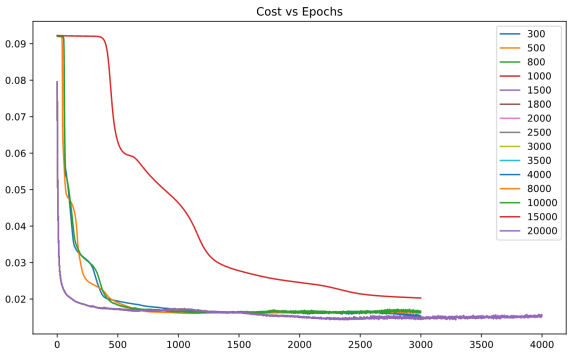


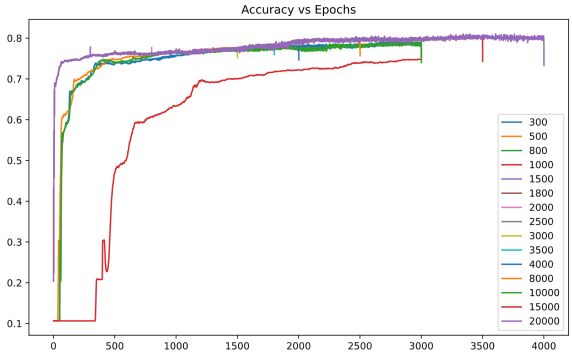
- With different activations





- With different epochs





Layer Dims	η	Epochs	Activations	GD Variant	Batch Size	Training Acc	Test Acc
[6, 64, 10]	0.015	50	TanH	SGD		0.741	0.723
[6, 64, 10]	0.01	15000	LReLu	MBGD	128	0.773	0.773
[6, 64, 10]	0.15	350	TanH	MBGD	128	0.77	0.752
[6, 64, 10]	0.02	50000	LReLu	MBGD	256	0.756	0.77
[6, 64, 10]	0.1	15000	LReLu	MBGD	256	0.81	0.745
[6, 64, 16, 10]	0.001	100	ReLu	SGD		0.758	0.755
[6, 64, 32, 10]	0.01	60	TanH	SGD		0.758	0.748
[6, 64, 32, 10]	0.13	3000	LReLu	MBGD	64	0.786	0.753
[6, 64, 32, 10]	0.13	2500	ReLu	MBGD	64	0.795	0.757
[6, 64, 32, 10]	0.13	3000	TanH	MBGD	64	0.784	0.758
[6, 64, 32, 10]	0.1	7000	LReLu	MBGD	128	0.89	0.703

Table 1: Performance of models of various architectures and hyperparameters.
(η = Learning Rate, LReLu = Leaky ReLu, GD = Gradient Descent, MBGD = Mini Batch Gradient Descent, SGD = Stochastic Gradient Descent)