Pattern Recognition & Machine Learning

Lab-7

“Multi-Layer Perceptrons, K-Means Clustering and Neural Networks”

Objectives

Following were the Objectives related to Fulfillment of Lab-7 :-

1. Implement Multi-Layered Perceptron from Scratch and then compare it with Sklearn’s Implementation, under certain conditions.

2. Implement K-Means Clustering from Scratch, vary the number of clusters, display the centroids and then compare it with Sklearn’s Implementation.

3. Although as an Optional Question, Implement Neural Network for MNIST Dataset, using PyTorch implementation and observe the change in results under certain conditions.

Datasets

Following were the Datasets, used in Lab-7 :-

1. Wheat Seed Dataset :- Download Wheat Seeds Dataset
2. MNIST Dataset :- Using Sklearn’s (fetch OpenML) and Pytorch’s (Dataset)

Dependencies

1. Numpy
2. Pandas
3. Matplotlib
4. Seaborn
5. Sklearn
6. Pytorch

Preprocessing Methods

1. For easier handling of Dataset and better runtime wise implementation, Datasets were required to be converted into Numpy Arrays/Matrices. As Pandas is not computationally efficient, compared to Numpy Implementations, Wheat Seed Dataset and Sklearn's MNIST Dataset were converted .

2. For better performance and calculations,Wheat Seed Dataset was standardized and MNIST Dataset was Feature-Scaled, as maximum value was 255.0 and minimum was 0.0 . Adjusting values between 0 and 1 made things easier for Scratch Implementation.

Multi-Layered Perceptron

Procedures

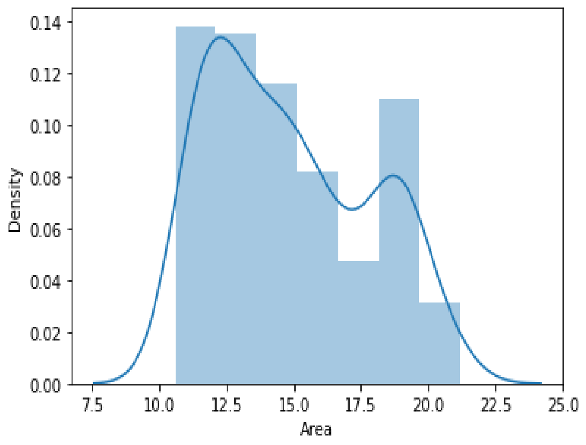
1. Understood the randomness of Distributions for various features in Wheat Seed Dataset. In addition to it, Target Classes were also witnessed.

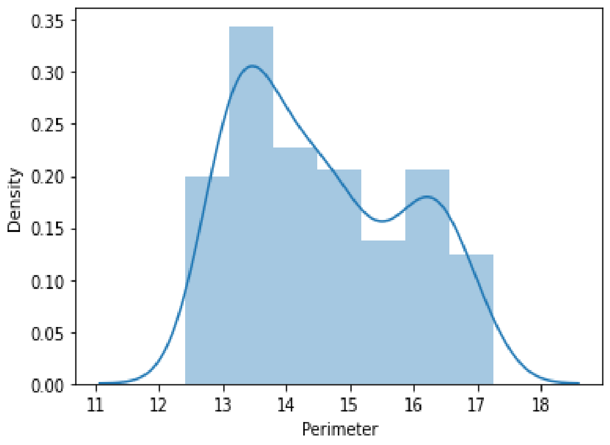
2. Followed a Pipeline for Scratch's Implementation, with few changes in workflow and conditions

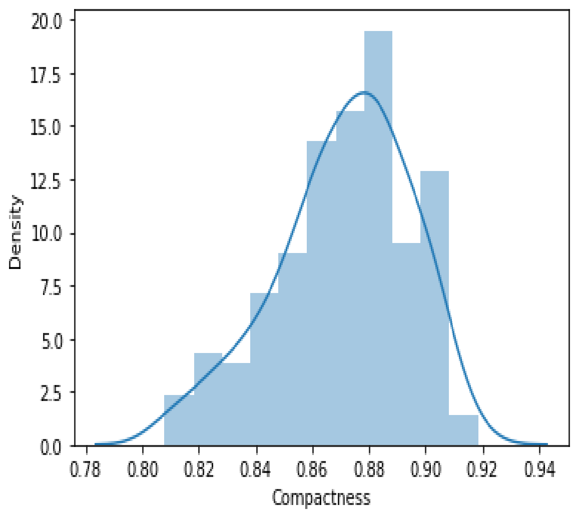
3. Calculated Loss and Test Accuracy corresponding to each implementation and for better understanding plotted it, for 100 different train-test splits.

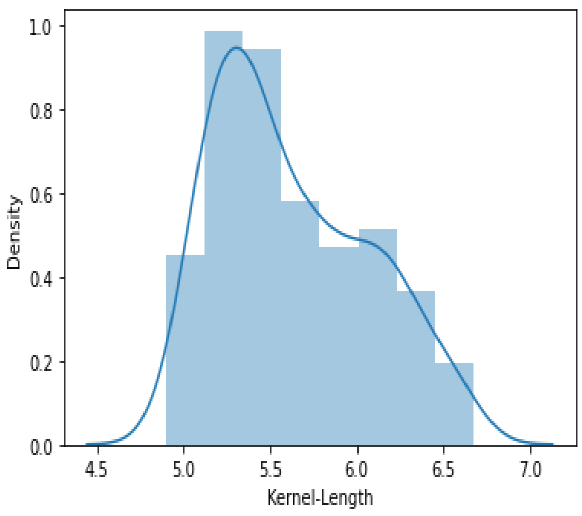
Feature-wise Distributions

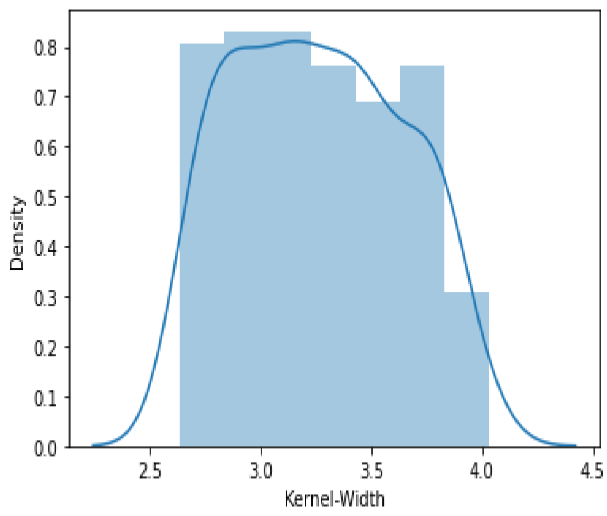
Following are the distributions, associated with Features of Wheat Seed Dataset, as following :-

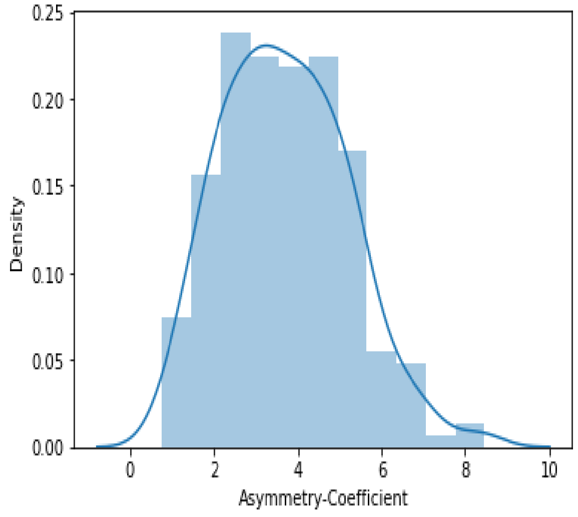


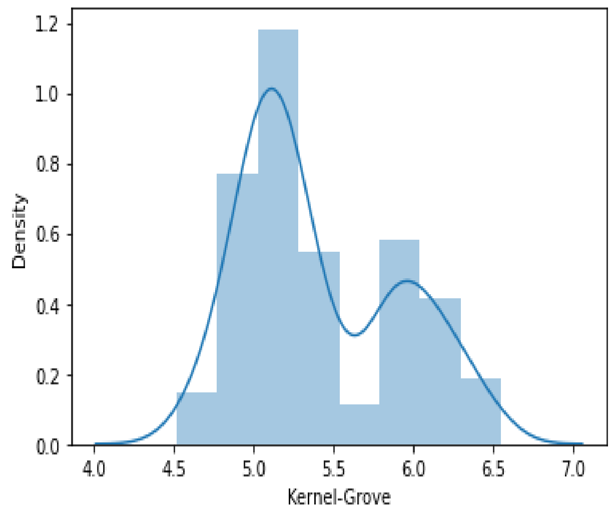


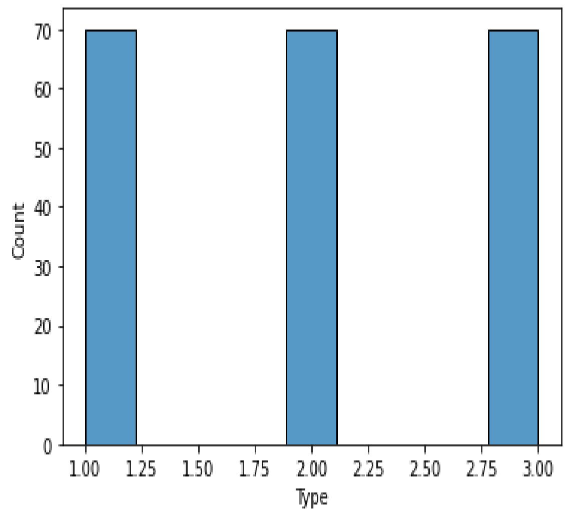

Density
Compactness
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
0.78 0.80 0.82 0.84 0.86 0.88 0.90 0.92 0.94


Density
Kernel-Length
0.0 0.2 0.4 0.6 0.8 1.0
4.5 5.0 5.5 6.0 6.5 7.0


Density
Kernel-Width
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
2.5 3.0 3.5 4.0 4.5

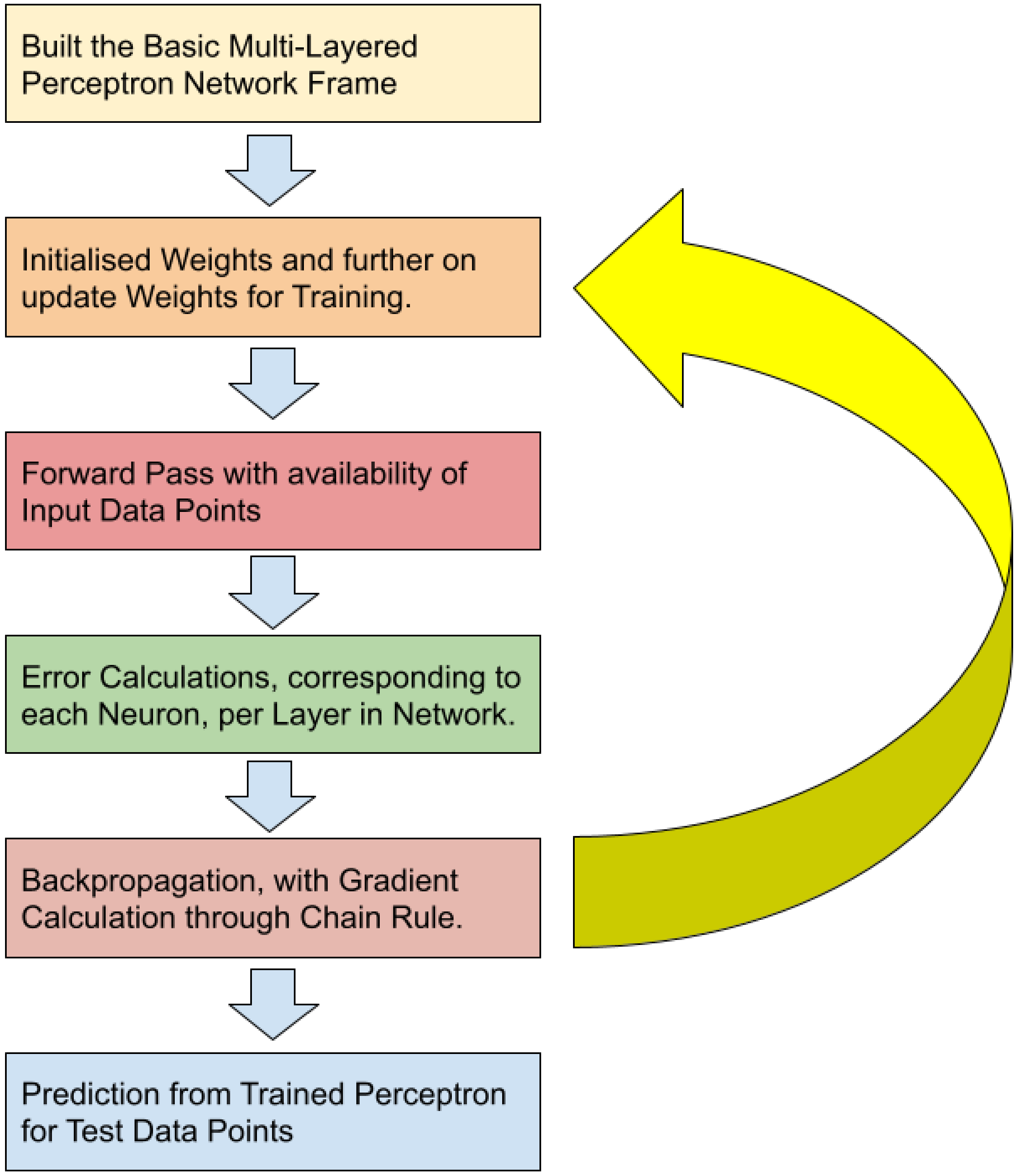

Density
Asymmetry-Coefficient
0.00 0.05 0.10 0.15 0.20 0.25
0 2 4 6 8 10


Density
Kernel-Grove
0.0 0.2 0.4 0.6 0.8 1.0 1.2
4.0 4.5 5.0 5.5 6.0 6.5 7.0


Count
Type
0 10 20 30 40 50 60 70
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00

Multi-Layered Perceptron Pipeline

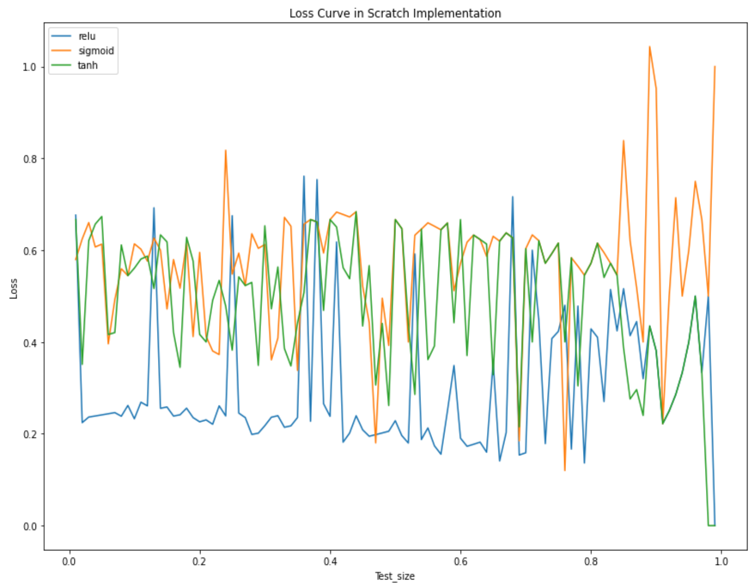
Following is the Pipeline for the Scratch Implementation of Multi-Layered Perceptron :-

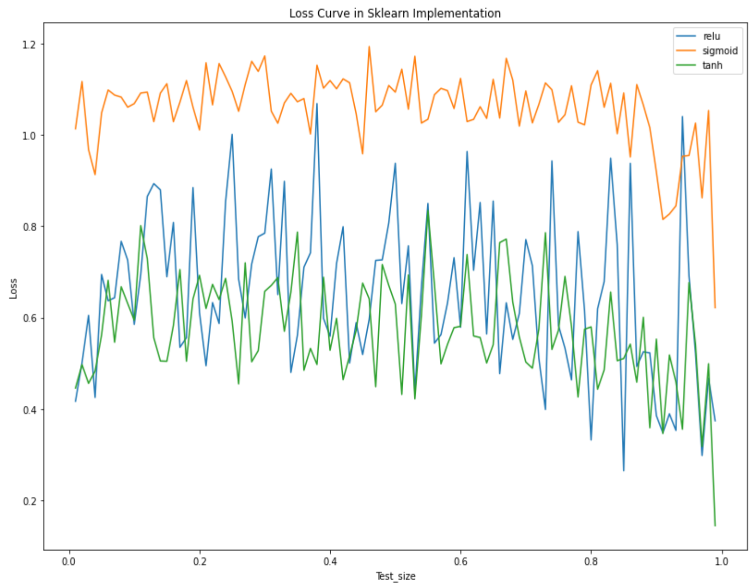

Built the Basic Multi-Layered Perceptron Network Frame
Initialised Weights and further on update Weights for Training.
Forward Pass with availability of Input Data Points
Error Calculations, corresponding to each Neuron, per Layer in Network.
Backpropagation, with Gradient Calculation through Chain Rule.
Prediction from Trained Perceptron for Test Data Points

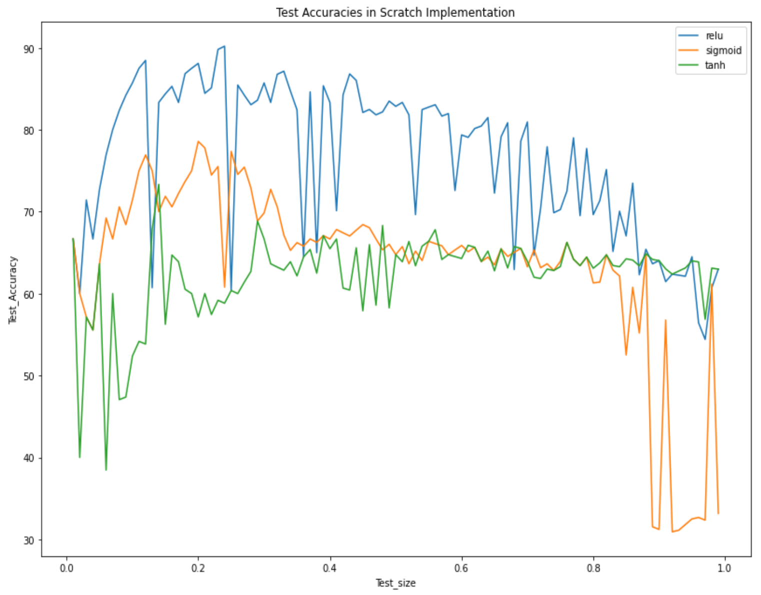
Comparison between Scratch and Sklearn Implementations

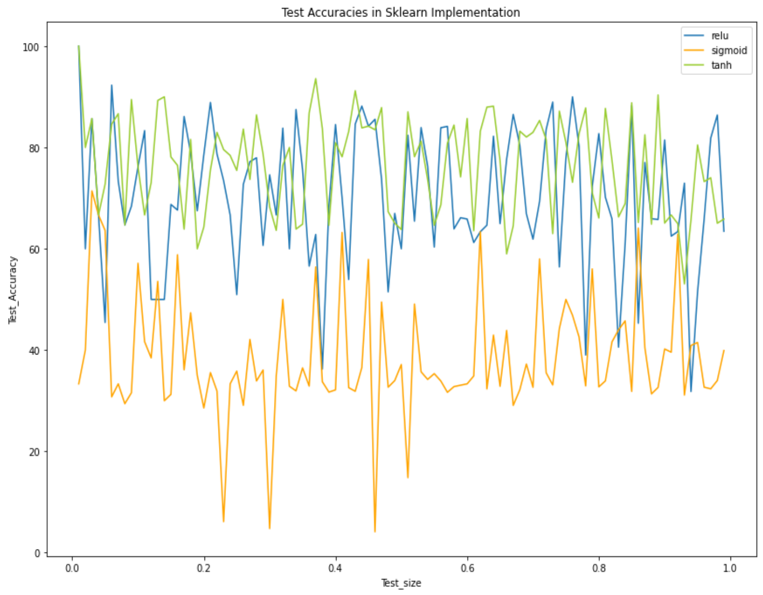
For the following configuration, the Comparison Graphs were plotted :-

1. Number of Epochs = 1000
2. Learning Rate = 0.001
3. Output Neurons = 3
4. Hidden Layers = 1
5. Number of Neurons in Hidden Layer = 7 (8 if bias neuron includes too)

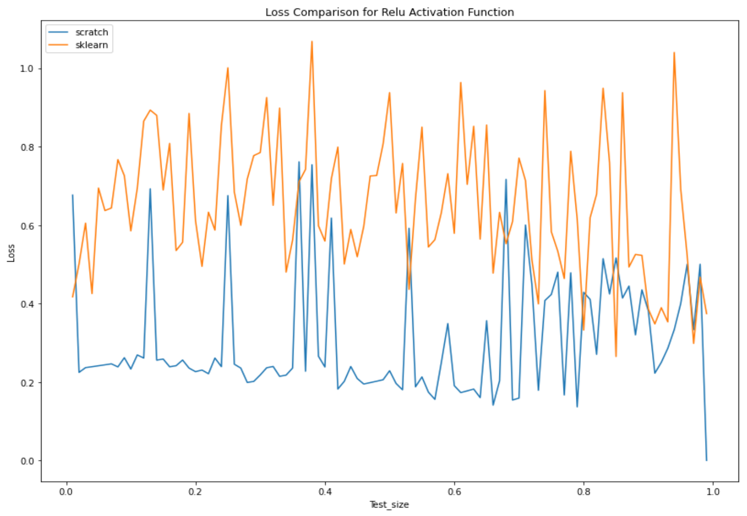


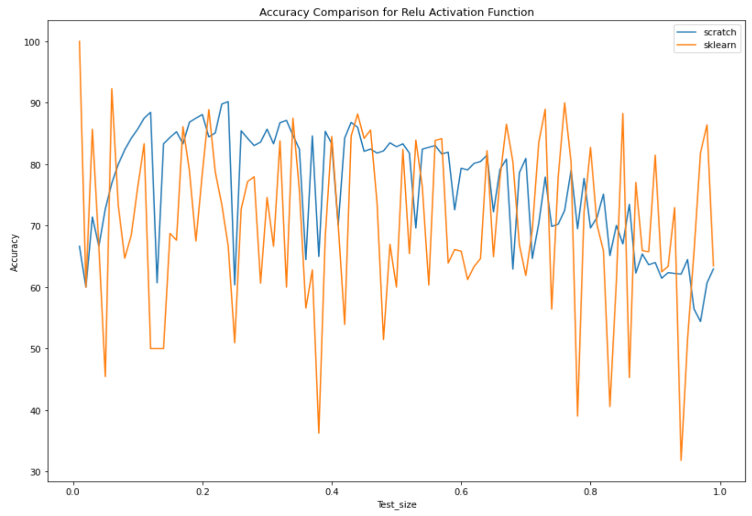


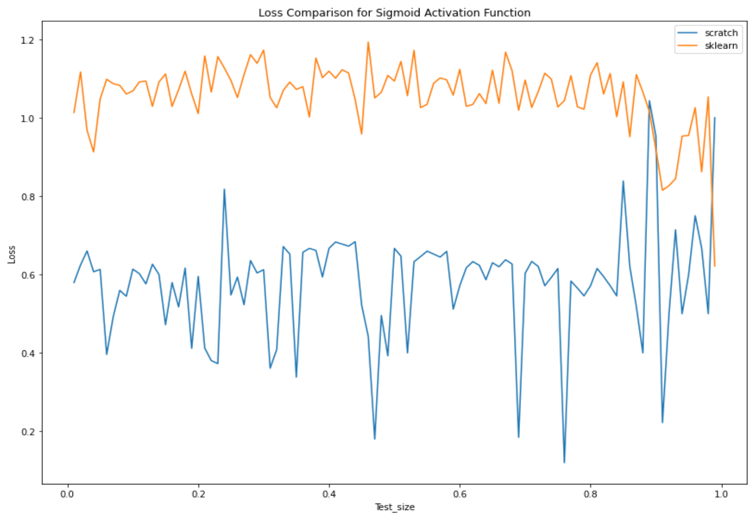


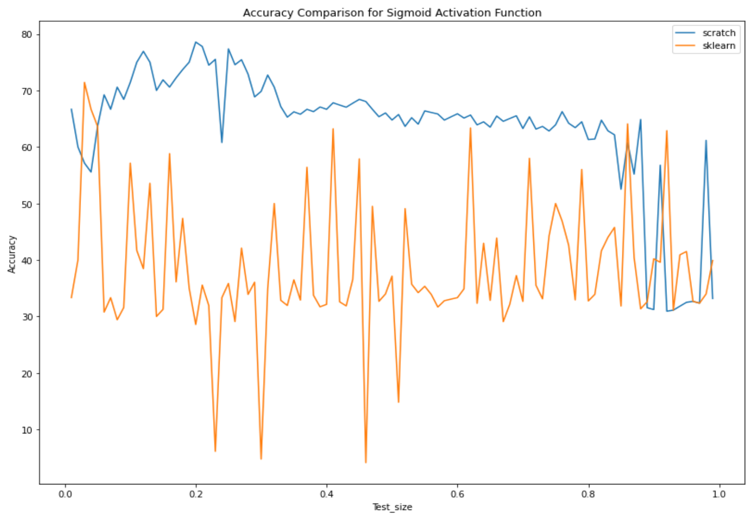


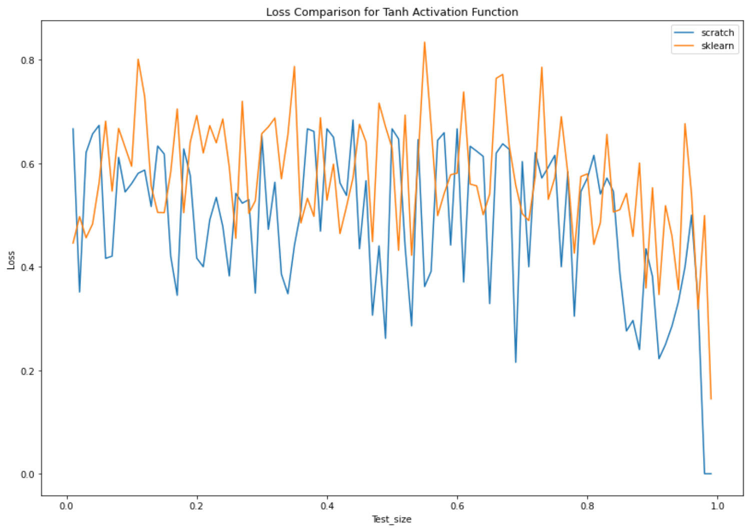
Also, Loss and Accuracy Curves were plotted for different Activation Functions, as following:-

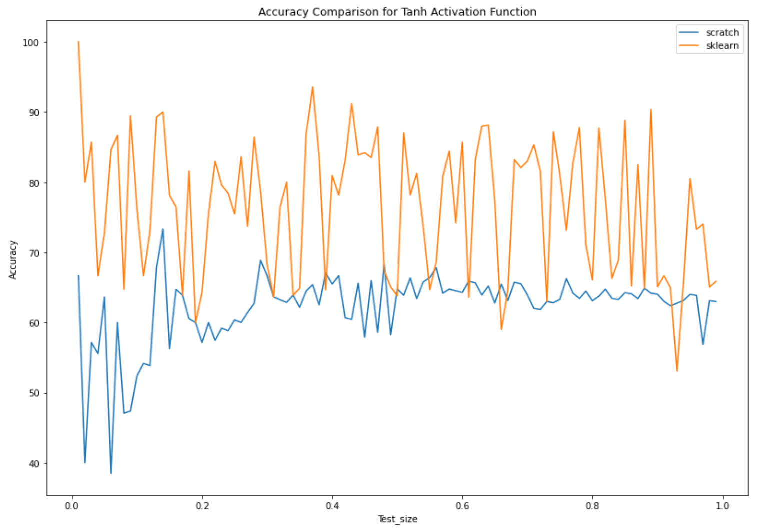








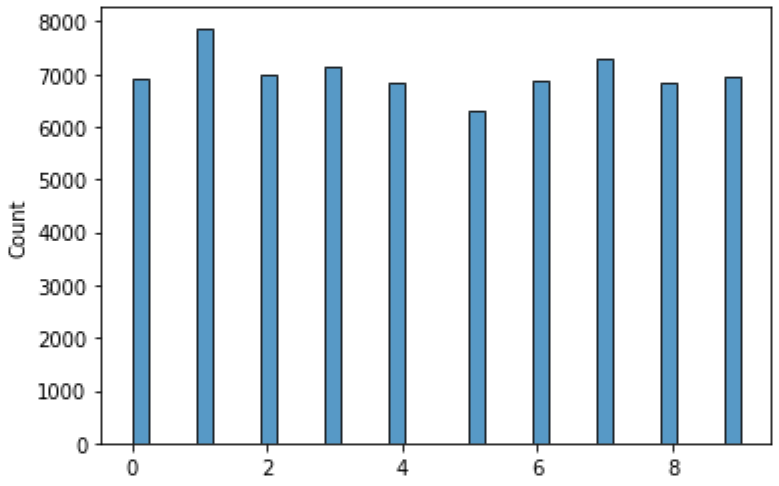




K-Means Clustering

Procedures

1. Understood the Distribution for Target Variable in MNIST Dataset, as following :-



2. Followed a Pipeline for Scratch's Implementation, with few variations in number of Clusters.

3. Calculated Test Accuracy corresponding to each implementation and for better understanding plotted the Centroids, for different numbers of clusters.

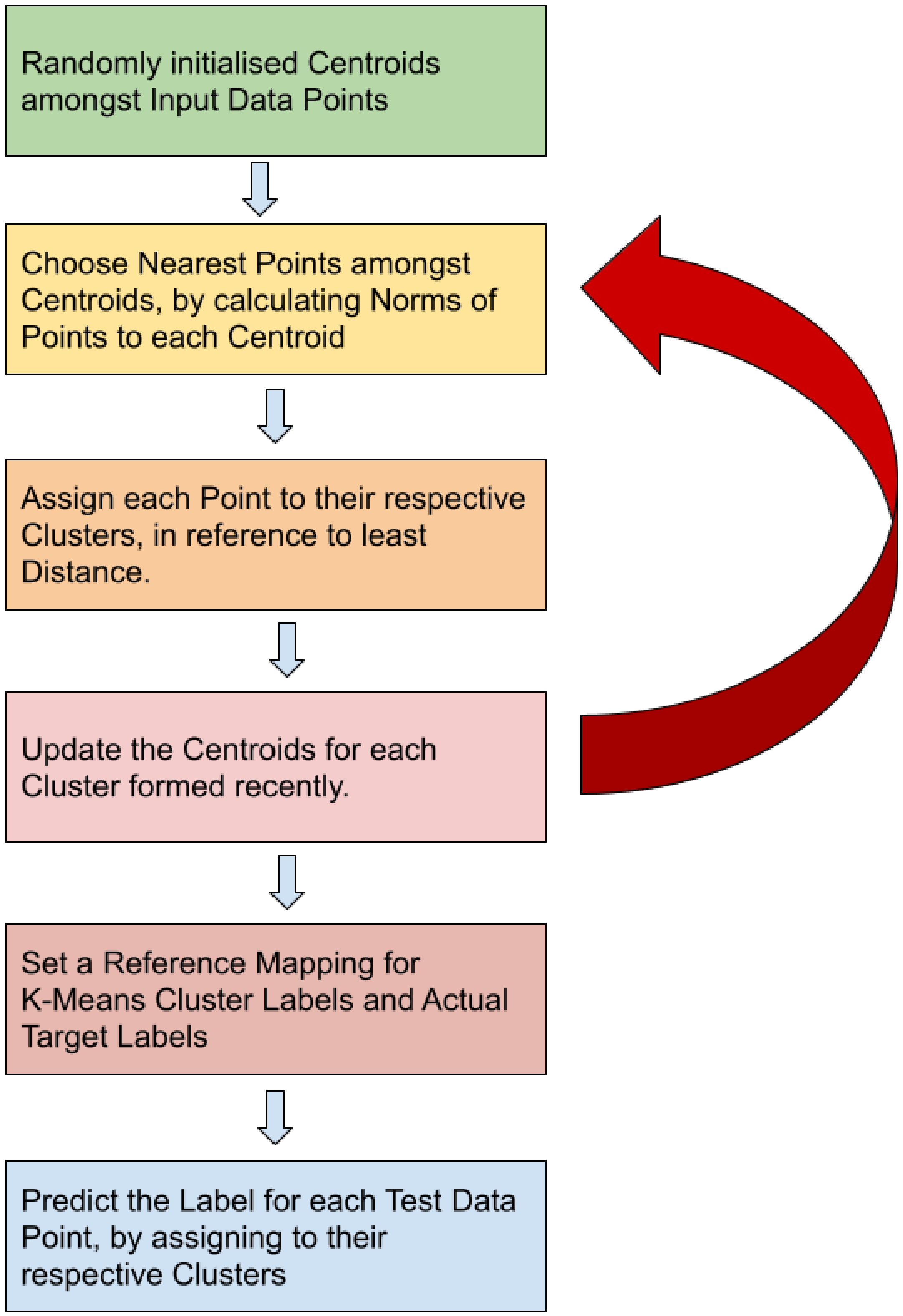
Euclidean Distance for Multi-Featured Data

Traditionally, Euclidean Distance considers 2 points, either 1-D,2-D or 3-D. Further on, the imagination doesn't let us think for multi-dimensional Euclidean Distance. But, instead of Euclidean Distance, Norm of 2 different Vectors of n-dimensions can be taken to calculate distance, as following :-

$$d_2(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2 = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

K-Means Clustering Pipeline

Following was the pipeline, incorporated for implementing K-Means Clustering :-

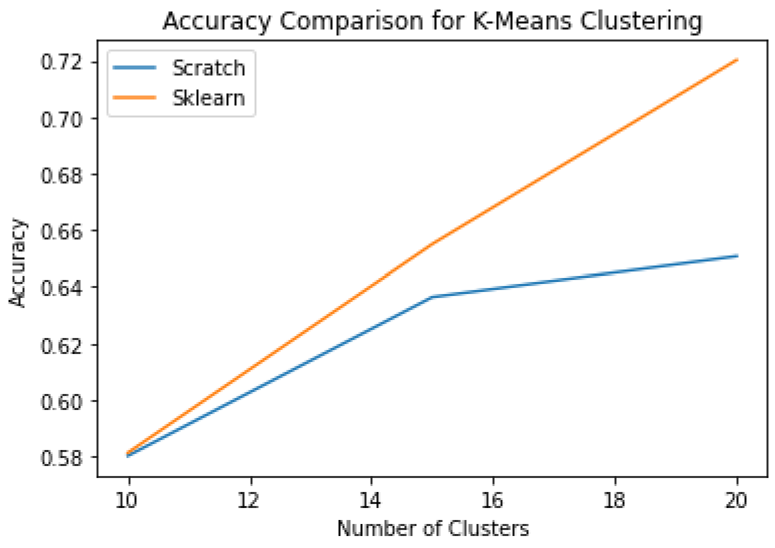

Randomly initialised Centroids amongst Input Data Points
Choose Nearest Points amongst Centroids, by calculating Norms of Points to each Centroid
Assign each Point to their respective Clusters, in reference to least Distance.
Update the Centroids for each Cluster formed recently.
Set a Reference Mapping for K-Means Cluster Labels and Actual Target Labels
Predict the Label for each Test Data Point, by assigning to their respective Clusters

Comparison between Scratch and Sklearn's Implementation

Following are some of the Graphs, indicating the differences between Scratch and Sklearn Implementation, with number of clusters = [10,15,20]

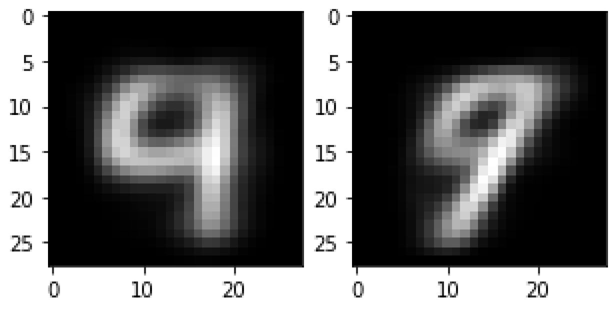
(Less Number of Clusters due to Runtime Complications) :-

1. Accuracy Curve with variation in number of Clusters

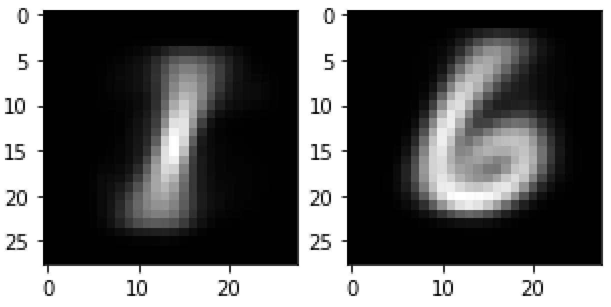


2. Centroids for each Cluster, at n_clusters = 10, for Scratch and Sklearn's Implementation respectively :-

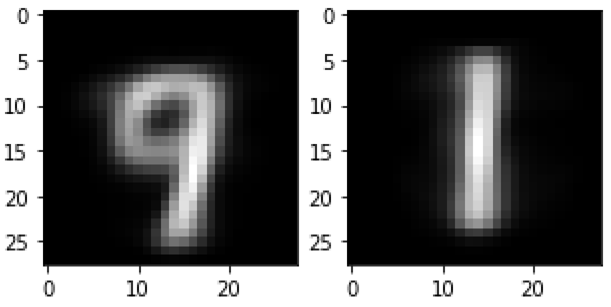
   Cluster Label-1 :-



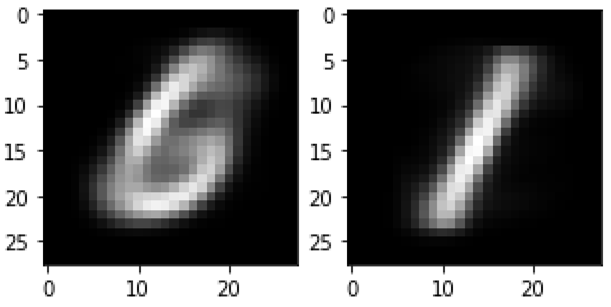
   Cluster Label-2 :-



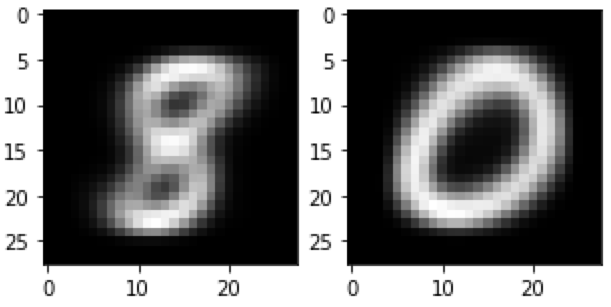
Cluster Label-3 :-



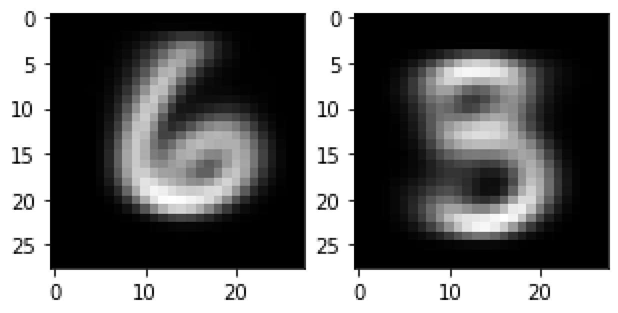
Cluster Label-4 :-



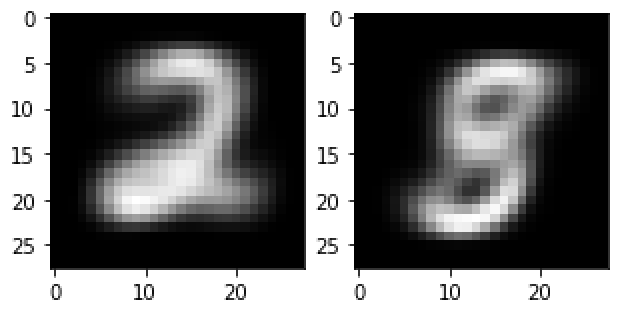
Cluster Label-5 :-



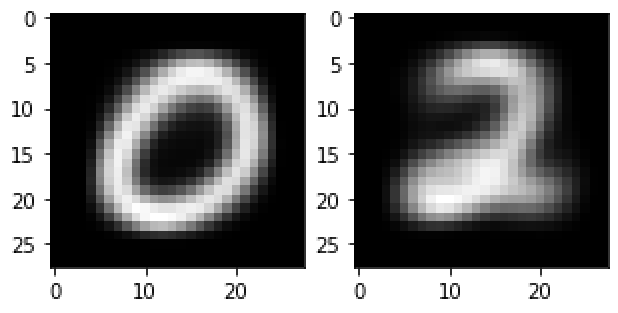
Cluster Label-6 :-



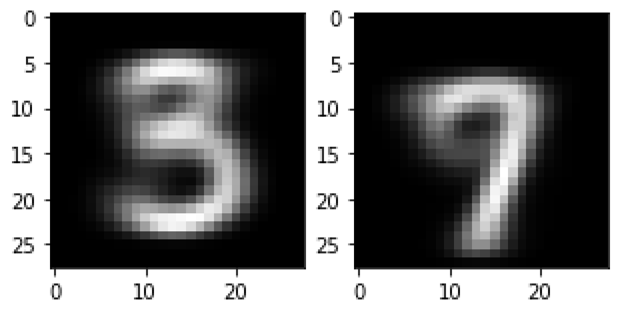
Cluster Label-7 :-



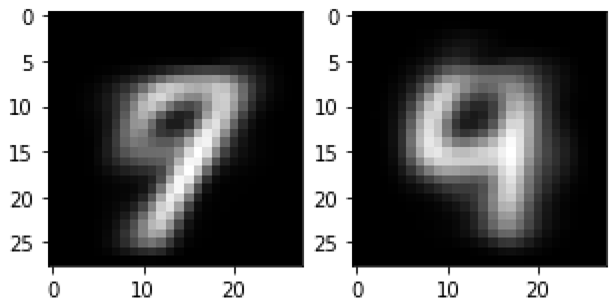
Cluster Label-8 :-



Cluster Label-9 :-



Cluster Label-10 :-



3. Reference Mapping for n_cluster = 10 in Scratch's Implementation, was as following :-

{0: 9, 1: 1, 2: 9, 3: 6, 4: 8, 5: 7, 6: 2, 7: 0, 8: 3, 9: 9}

Neural Networks

Procedures

1. Normalized Data Points in MNIST-Dataset, enabled Shuffle and chose batch_size as 64.

2. Neural Networks were built, according to the following parameter variations :-

    1. Number of Layers - [1,2,3,4]
       (epochs=15 and n_neuron = 392)

    2. Learning Rate - [0.0025,0.005,0.0075,0.01]
       (epochs=15 and n_neuron = 392)

    3. Momentum - [0.7,0.75,0.8,0.85]
       (epochs=15 and n_neuron = 392)

    4. Number of Neurons - [49,98,196,392]
       (epochs=15 and n_neuron = 392)

    5. Change in Loss Function - NLLL and Cross-Entropy
       (epochs=15 and n_neuron = 392)

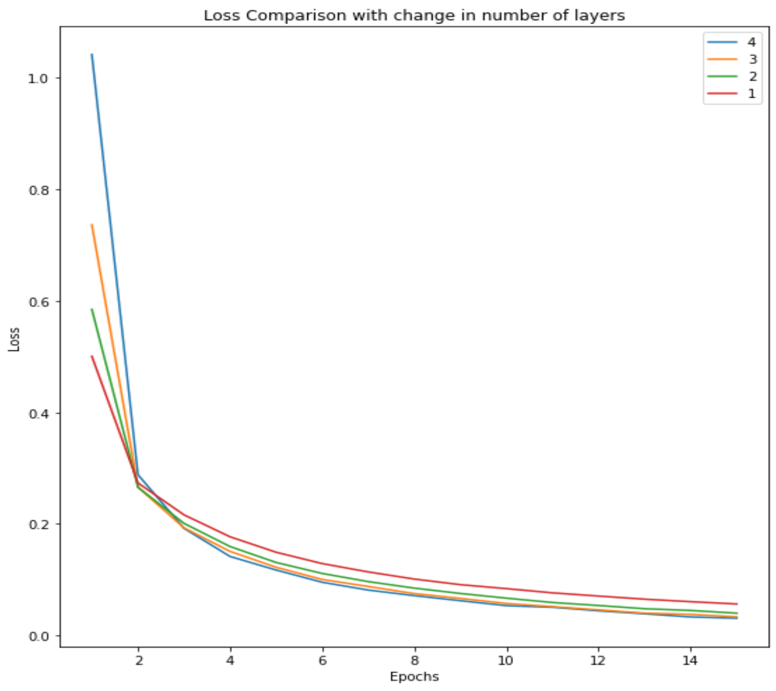
    6. Number of Epochs - [5,10,15,20]
       (n_layer=4 and n_neuron = 392)

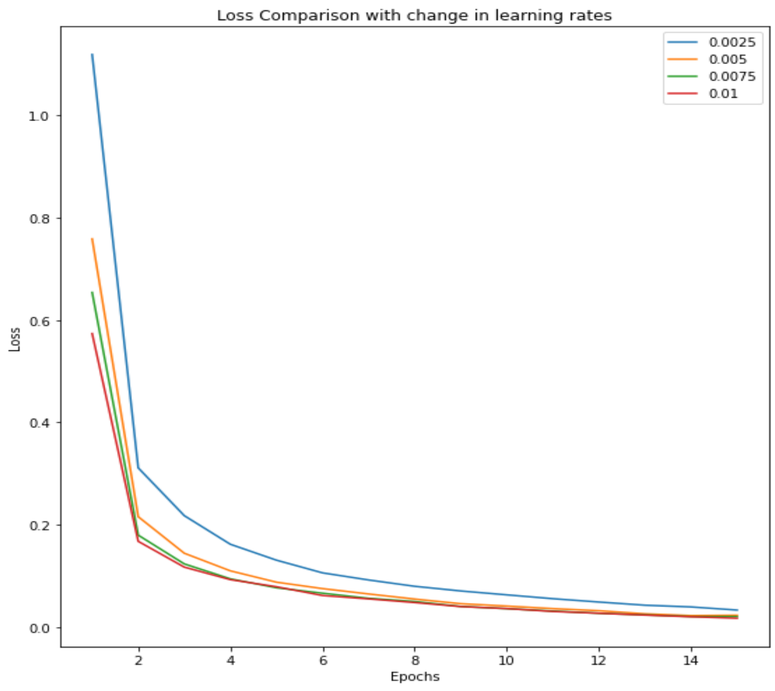
3. Computed Loss and Accuracy, corresponding to each Variety.

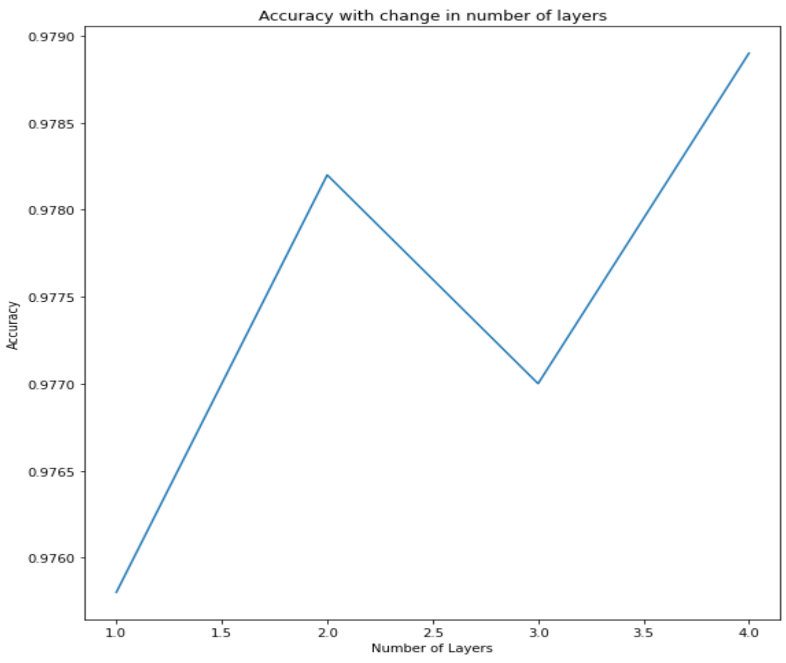
Comparative Plots for each parameter varieties

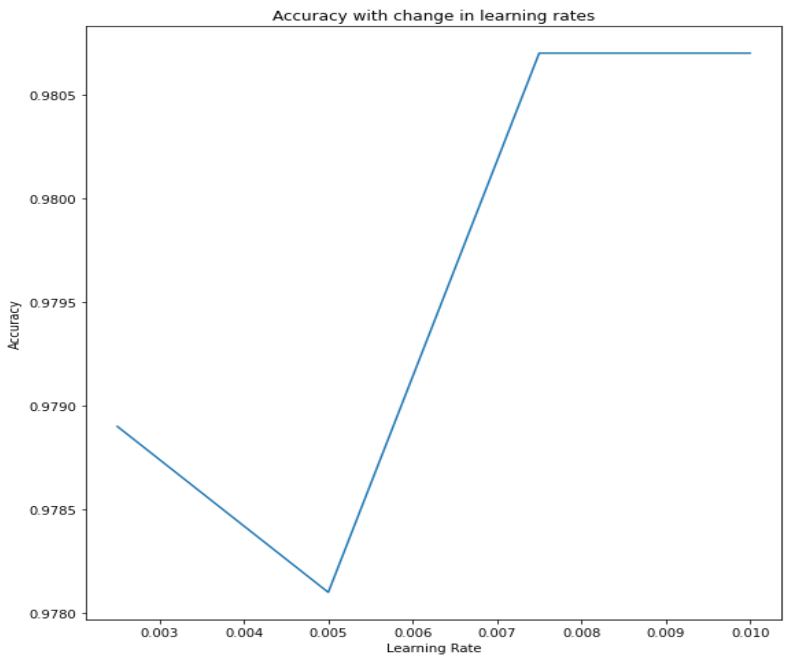
Following were the Loss and Accuracy plots for each parameter combinations associated with Neural Network :-

1. Plots according to variation in Number of Layers and Learning Rate

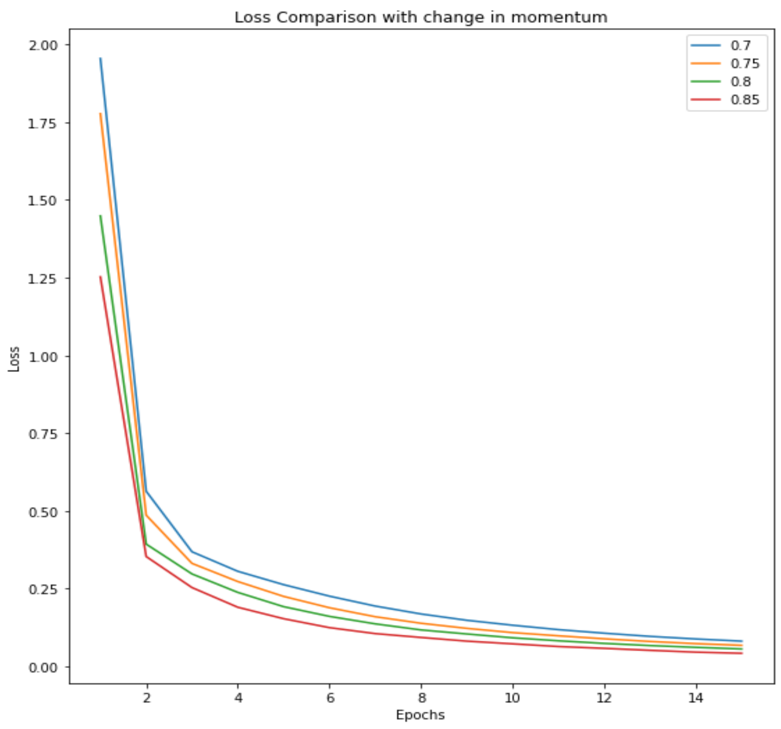


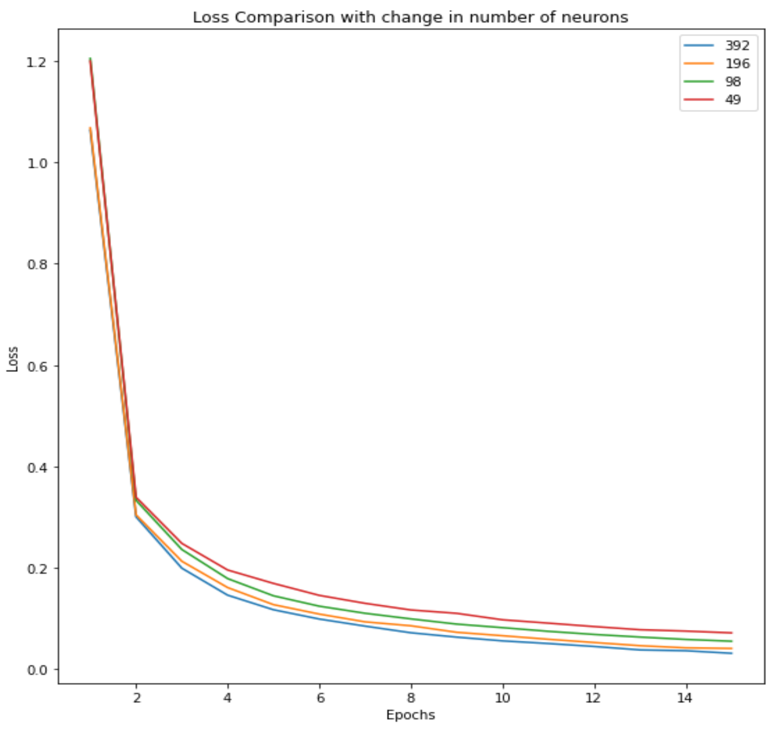


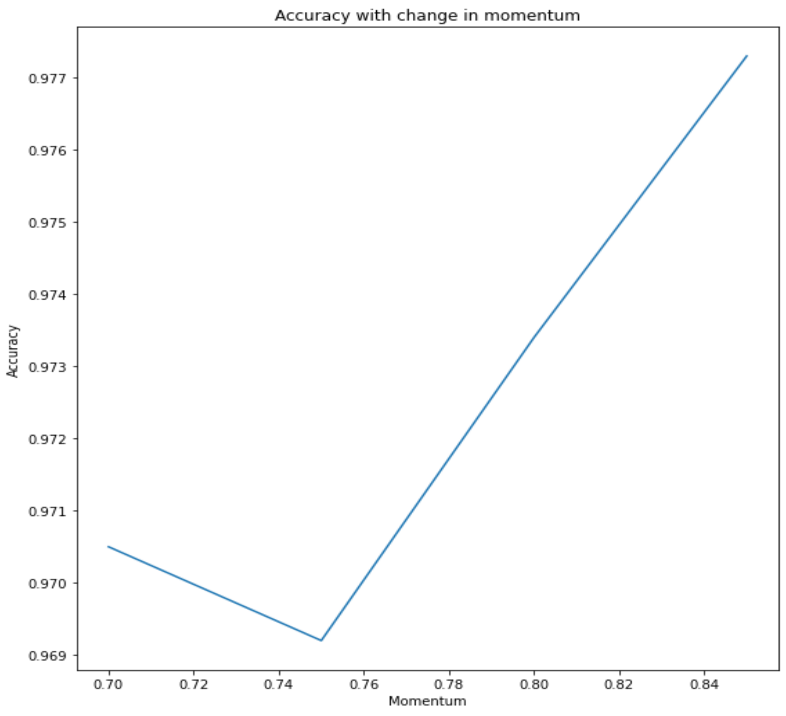


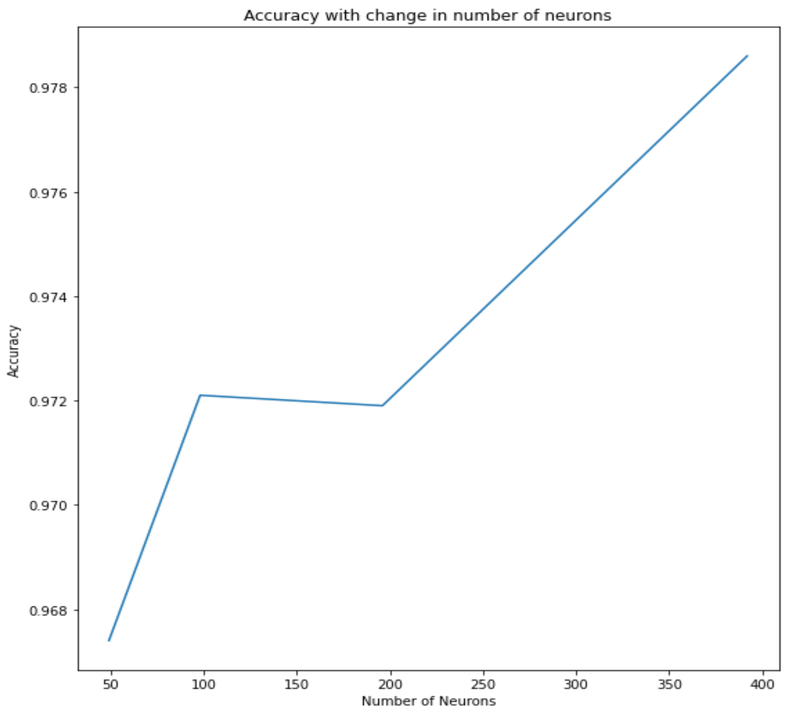


2. Plots according to change in Momentum and Number of Neurons

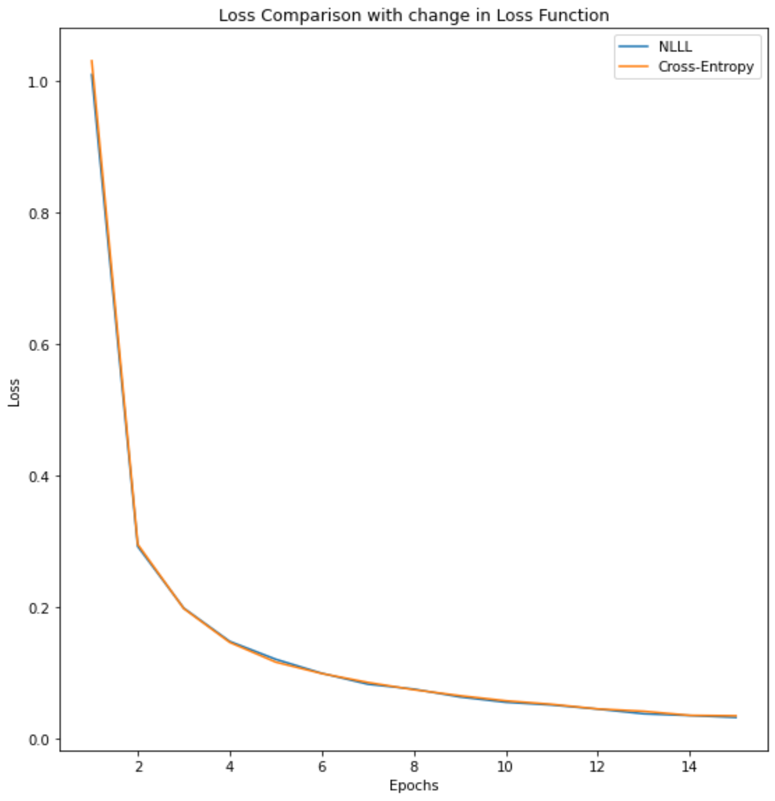


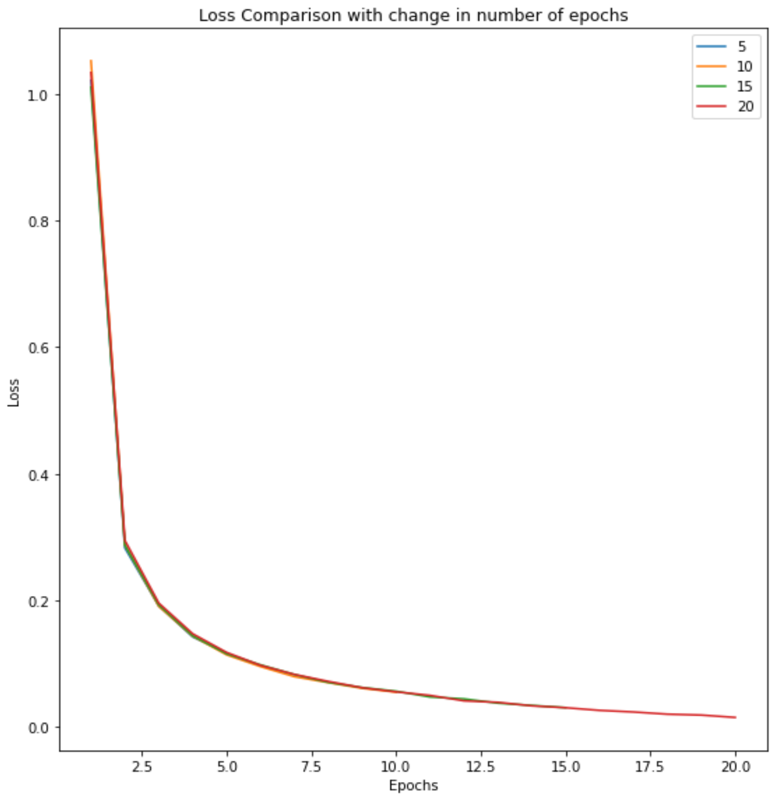


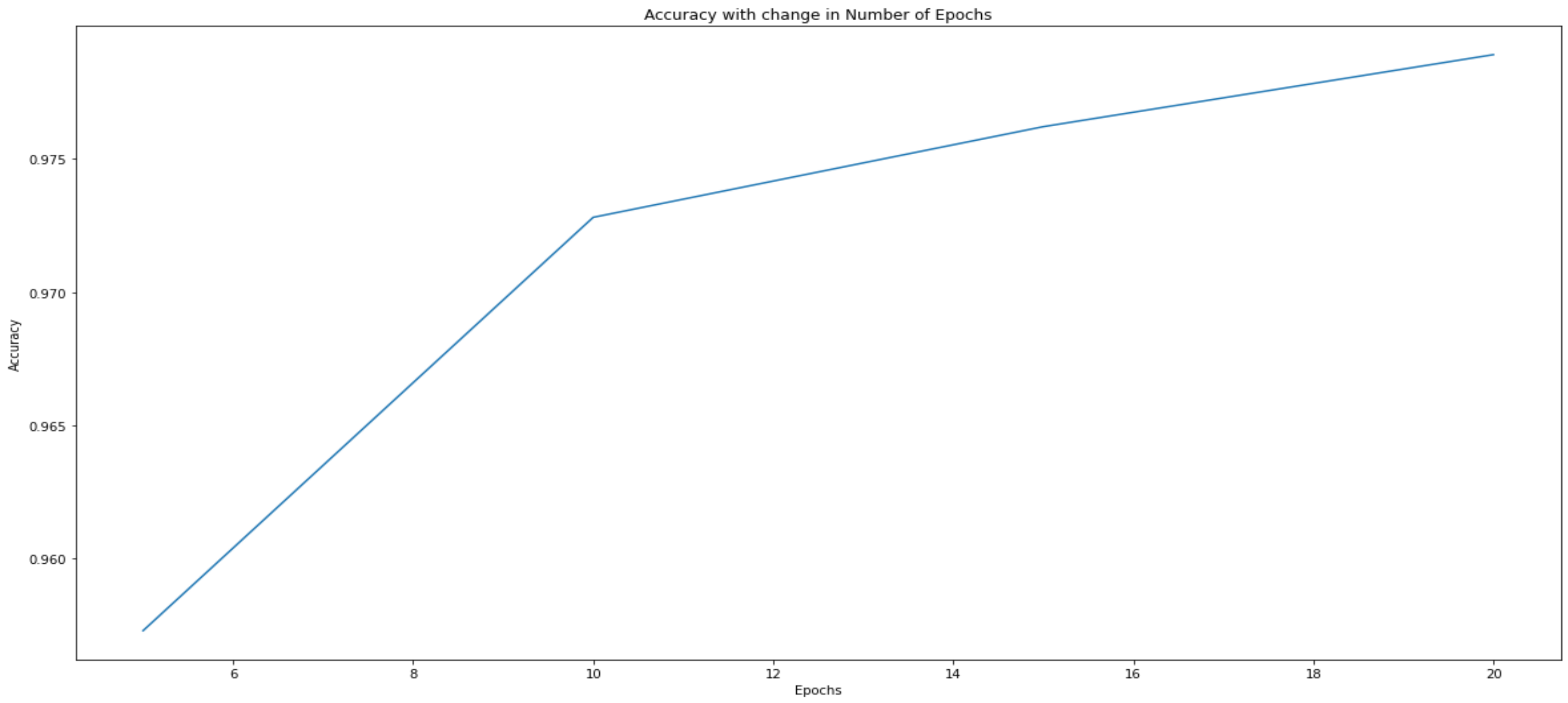




3. Plots according to change in Loss Functions and Number of Epochs







Conclusion

Following were the conclusions, regarding the above implementations :-

1. Differences in Loss and Accuracy for Scratch and Sklearn's implementation arose due to difference in configurations between scratch and sklearn's default Commands (Optimized with other additional parameters). This resulted in differences in Run-Time in a drastic fashion (like 20-30 time manifold).

2. According to different train-test splits, implementations don't improve or degrade always with increase/decrease in test_size. A peak can be located in between, same case for rock point too.

3. Before Normalizing the Values of Wheat Seed Dataset, the Loss for Sigmoid and Tanh Activation Functions were considerably high and Accuracies were around 30-40%. But after Normalizing the Values, Loss came down and the peaks for Accuracy curve went upto 70-90%.

4. For Neural Networks, change in Loss Function didn't affect the results that much significantly. Elsewhere, the results improved with improvements in learning rate, number of epochs, number of layers and momentum.