Classification of Fashion MNIST Data using Neural Networks

Gaurav Madhav Pai
Department of Computer Science
University at Buffalo (SUNY)
Buffalo, NY 14226 gmadhavp@buffalo,edu

Abstract

In this project, variations of neural networks are used to classify images in Fashion MNIST Dataset with 10 Classes. The three kinds of models are: Neural Networks designed from scratch, Neural Network Implementation using Keras and Convolution Neural Networks using Keras. The results of these models are compared and discussed.

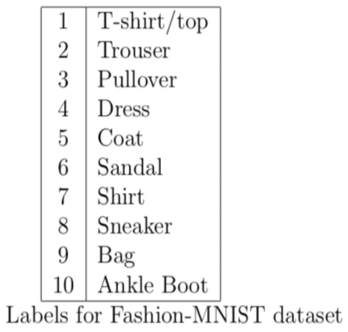
1. Introduction

A Neural Network works based on the Universal Approximation Theorem. In this project performance of classification of three different models have been done on a sample of 60000 training images spread across explained in the next section. The evaluation of performance of three models has been done using a testing set with 10000 unseen images. The following models were used for classification

1. Neural Network designed from scratch with 300 nodes in the hidden layer
2. Neural Network using Keras with 3 Hidden Layers
3. Convolution Neural Network using Keras with Convolution Layer, MaxPooling Layers

2. Dataset

For training and testing of our classifiers, we will use the Fashion-MNIST dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.



1	T-shirt/top
2	Trouser
3	Pullover
4	Dress
5	Coat
6	Sandal
7	Shirt
8	Sneaker
9	Bag
10	Ankle Boot

Labels for Fashion-MNIST dataset

Figure 1: List of Classes in the dataset

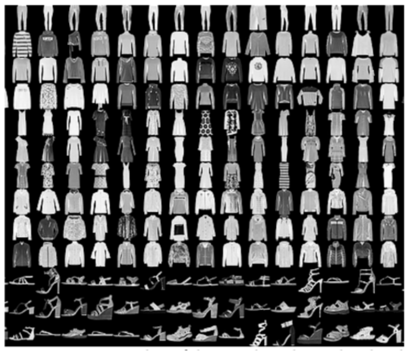
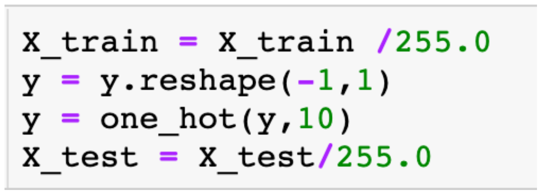


Figure 2: Example of how the data looks like.

3. Pre-processing

Since, the target variable has values have a single value between 0 and 9 representing the 10 classes, one hot encoding is used to convert the single value into a one hot vector with 10 values.
Further since the pixel values of each image is between 0 and 255, it is normalized by dividing each pixel by 255 to bring the values down to the range 0 and 1.



```
X_train = X_train /255.0
y = y.reshape(-1,1)
y = one_hot(y,10)
X_test = X_test/255.0
```

Figure 3 – Code for normalization and one hot encoding

4. Architecture

4.1 Activation Functions

1. In **Sigmoid Activation Function,** we pass the linear equation through a sigmoid function in order to squeeze the output of the linear equation between 0 and 1. This activation function applied to all the layers but the final output layer. The linear equation that incorporates the weights and bias to give a prediction is given in Figure 4. Figure 5 shows the Sigmoid function

$$Z = WX + B$$

Figure 4 – Genesis Equation

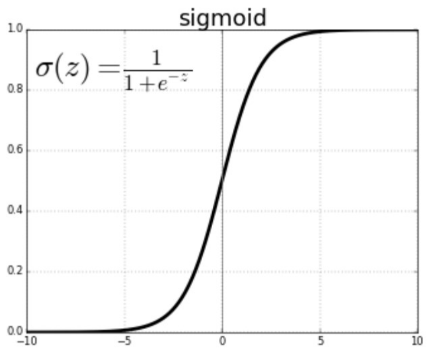


Figure 5 - Sigmoid Function

2. **The SoftMax function** is used in the output layer, which gives out values as a vector whose length is the number of classes in the dataset. The sum of all the values of the output vector add up to 1 and the class with the highest value (probability) is considered the prediction. The following figure gives the formula for the SoftMax activation function.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

Figure 6 – SoftMax Function

4.2 Loss Function

The loss function gives us a single number representing the how bad is the classification performance of the algorithm. Figure 6 represents the Categorical Cross Entropy loss function that is used by the Gradient Descent for optimization of weights and bias.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Figure 7 - Categorical Cross Entropy Loss Function

4.3 Neural Networks

A Neural Network consists of an input later some hidden layer and an output layer. Each layer consists of a neuron which compute the result of a linear equation (figure 4) and is followed by an activation function. The input enters from the input layers and goes through a hidden layers and arrives at the output layers as predictions.

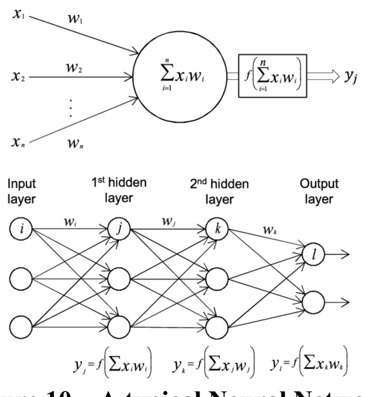


Figure 10 – A typical Neural Network

4.4 Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.
The architecture of a ConvNet is similar to that of the connectivity of neurons in the Human Brain and was inspired by the organization of the Visual Cortex. The image that enters the network typically goes through convolution layer then a pooling layer and fully connected layer to give out predictions.

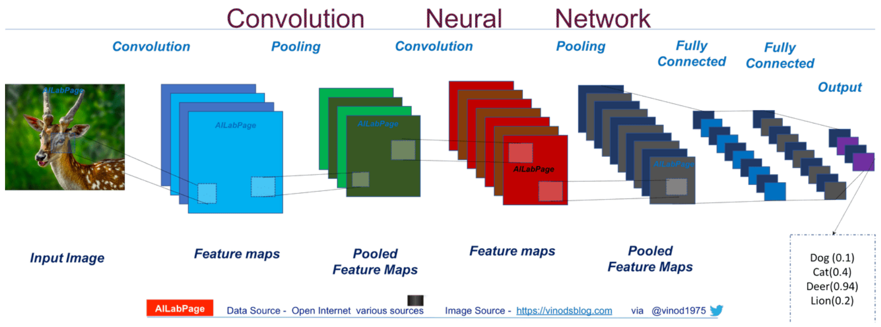


Figure 11 – A typical Convolutional Neural Network

4.5 Feedforward

The neural network essentially consists of 2 main steps. The first of which is feedforward. In this is part, the data makes a complete pass from the first layer to the last layer of the network to give a prediction.

```
def forward_pass(self, X):
    z1 = self.w1.dot(X.T)
    a1 = self.sigmoid(z1)
    z2 = self.w2.dot(a1)
    a2 = self.softmax(z2)
    return z1,a1,z2,a2
```

Figure 12 – Code showing the forward pass

4.6 Backpropagation

Backpropagation in neural network is the second step. After the forward pass, in order to improve the weights, backpropagation uses the Gradient Descent algorithm to optimize the weights and biases in the network. This process of feedforward and backpropagation is done until the networks gives a good result on the task it was intended to do. Backpropagation algorithm requires the gradients of the activation functions.

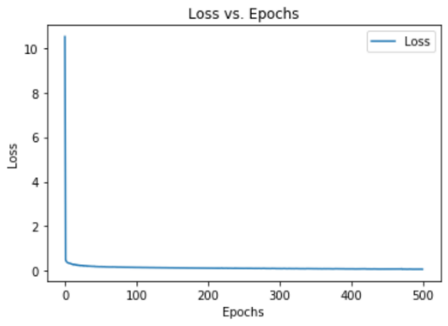
```
itermediate_1 = a2 - y_train.T
intermediate_2 = self.w2.T.dot(imtermediate_1) * self.sigmoid_prime(z1)
dw1 = intermediate_2.dot(X_train)
dw2 = intermediate_1.dot(a1.T)
self.w1 -= self.alpha * dw1
self.w2 -= self.alpha * dw2
```

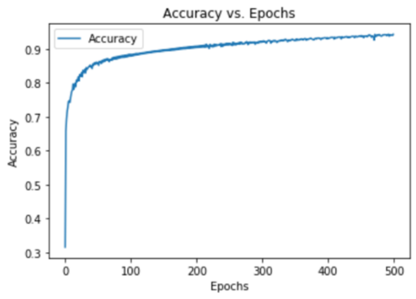
Figure 13 – Code showing the backpropagation step

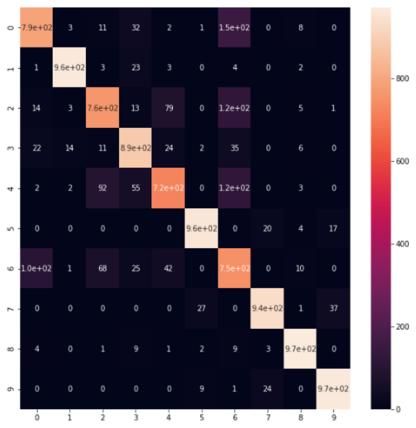
5. Results

5.1 Training the model

Neural Networks From Scratch: Epocs:500, Learning Rate : 10e-4, Batches : 15, Hidden Layer Nodes : 300

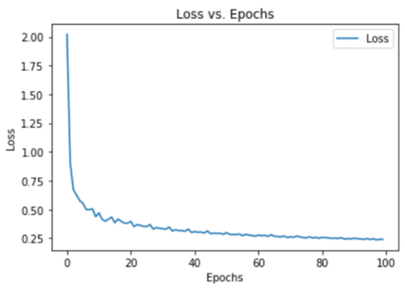


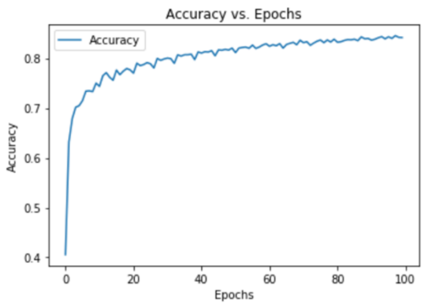


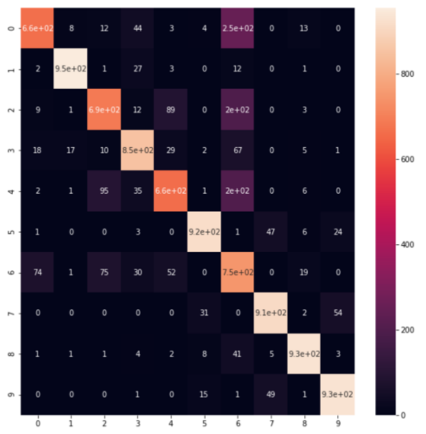


```
             precision    recall  f1-score   support

          0       0.84      0.79      0.82      1000
          1       0.98      0.96      0.97      1000
          2       0.80      0.76      0.78      1000
          3       0.85      0.89      0.87      1000
          4       0.83      0.72      0.77      1000
          5       0.96      0.96      0.96      1000
          6       0.63      0.75      0.68      1000
          7       0.95      0.94      0.94      1000
          8       0.96      0.97      0.97      1000
          9       0.95      0.97      0.96      1000

avg / total       0.87      0.87      0.87     10000
```

Neural Networks From Scratch: Epocs:100, Learning Rate : 10e-5, Batches : 15, Hidden Layer Nodes : 300

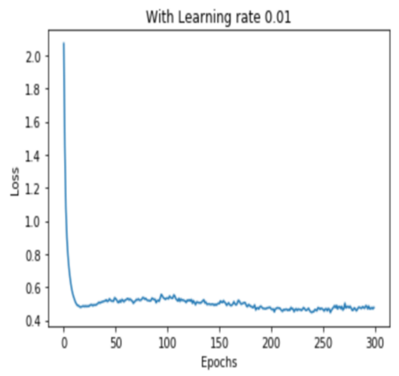


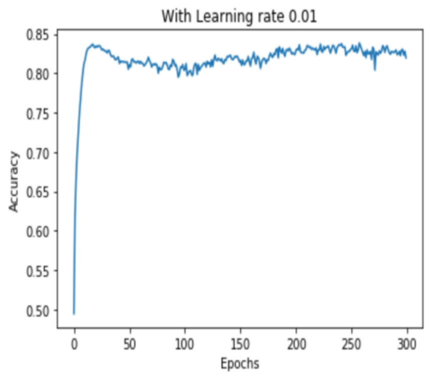




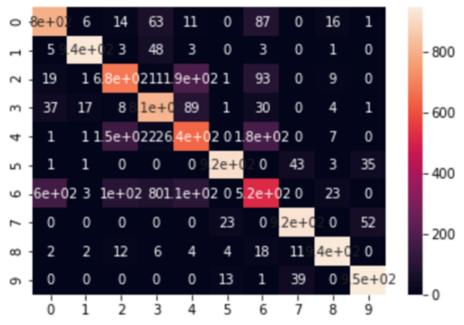
```
             precision    recall  f1-score   support

          0       0.86      0.67      0.75      1000
          1       0.97      0.95      0.96      1000
          2       0.78      0.69      0.73      1000
          3       0.85      0.85      0.85      1000
          4       0.79      0.66      0.72      1000
          5       0.94      0.92      0.93      1000
          6       0.49      0.75      0.59      1000
          7       0.90      0.91      0.91      1000
          8       0.94      0.93      0.94      1000
          9       0.92      0.93      0.93      1000

avg / total       0.84      0.83      0.83     10000
```

Neural Networks Keras : Epocs:200, Learning Rate : 10e-2, Hidden Layer Nodes : 300,200,75

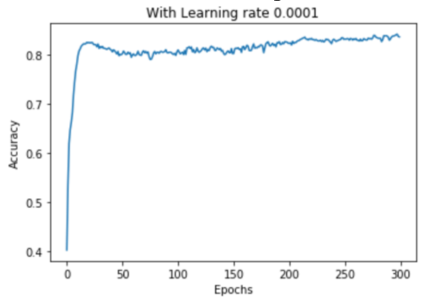


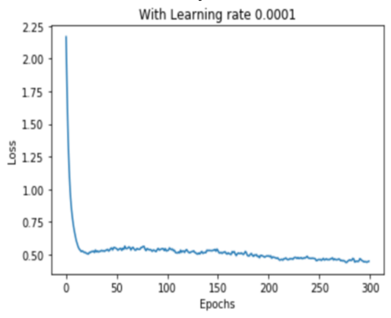


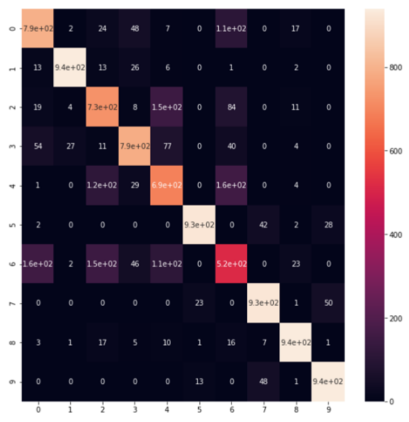
```
             precision    recall  f1-score   support

          0       0.78      0.80      0.79      1000
          1       0.97      0.94      0.95      1000
          2       0.70      0.68      0.69      1000
          3       0.78      0.81      0.80      1000
          4       0.61      0.64      0.63      1000
          5       0.96      0.92      0.94      1000
          6       0.56      0.52      0.54      1000
          7       0.91      0.93      0.92      1000
          8       0.94      0.94      0.94      1000
          9       0.91      0.95      0.93      1000

avg / total       0.81      0.81      0.81     10000
```



Neural Networks Keras : Epocs:200, Learning Rate : 10e-4, Hidden Layer Nodes : 200,100,50

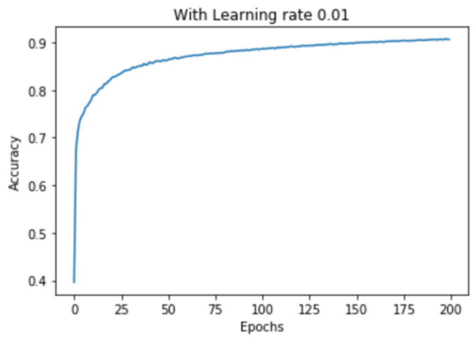


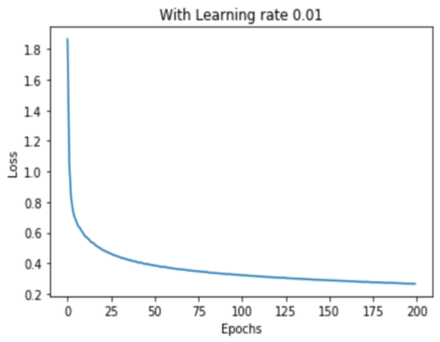




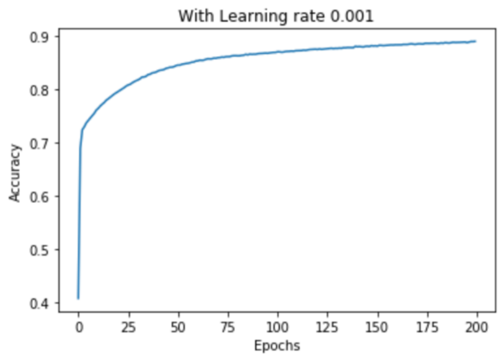
```
             precision    recall  f1-score   support

          0       0.76      0.79      0.78      1000
          1       0.96      0.94      0.95      1000
          2       0.69      0.73      0.71      1000
          3       0.83      0.79      0.81      1000
          4       0.66      0.69      0.67      1000
          5       0.96      0.93      0.94      1000
          6       0.56      0.52      0.54      1000
          7       0.91      0.93      0.92      1000
          8       0.94      0.94      0.94      1000
          9       0.92      0.94      0.93      1000

avg / total       0.82      0.82      0.82     10000
```

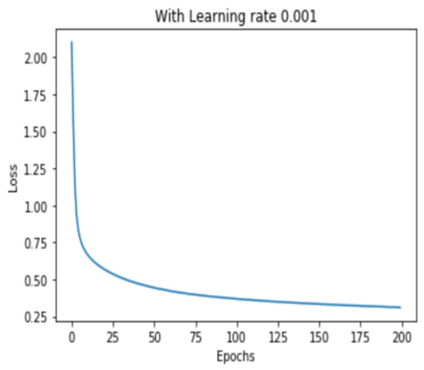
Convolutional Neural Networks Keras : Epocs:200, Learning Rate : 10e-2, Batches : 15, Hidden Layer Nodes : 200,100,50





Convolutional Neural Networks Keras : Epocs:200, Learning Rate : 10e-2, Batches : 15, Hidden Layer Nodes : 200,300





5.2 Testing the model

These are the results obtained on the test set

Neural Networks From Scratch: Epocs:500, Learning Rate : 10e-4, Batches : 15, Hidden Layer Nodes : 300 **Accuracy: 87%**

Neural Networks From Scratch: Epocs:100, Learning Rate : 10e-5, Batches : 15, Hidden Layer Nodes : 300 **Accuracy:82%**

Neural Networks Keras : Epocs:200, Learning Rate : 10e-2, Hidden Layer Nodes : 300,200,75 **Accuracy: 81.2%**

Neural Networks Keras : Epocs:200, Learning Rate : 10e-4, Hidden Layer Nodes : 200,100,50 **Accuracy: 81.8%**

Convolutional Neural Networks Keras : Epocs:200, Learning Rate : 10e-2, Batches : 15, Hidden Layer Nodes : 200,100,50 **Accuracy: 89.1%**

Convolutional Neural Networks Keras : Epocs:200, Learning Rate : 10e-2, Batches : 15, Hidden Layer Nodes : 200,300 **Accuracy: 88%**

5.3 Evaluation Metrics

This following Figure A give the formulae for computation of the evaluation metrics i.e Accuracy, Precision and Recall.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Figure A : Accuracy , Precision and Recall

6. Conclusion

Looking at the results, it is concluded that Convolutional Neural Networks with the following configuration performs the best among the models.

Convolutional Neural Networks Keras : Epocs:200, Learning Rate : 10e-2, Batches : 15, Hidden Layer Nodes : 200,100,50 **Accuracy: 89.1%**