Classify Fashion-MNIST clothing images using Neural Network

Shivaprakash Rajshekar Hiremath
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
hiremath@buffalo.edu

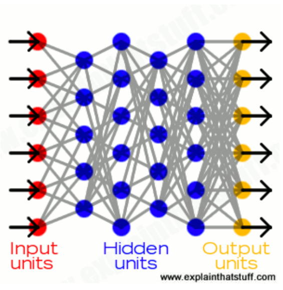
Abstract

In this project, we study and build a neural network and convolutional neural network model for a ten-class classification problem. We are using Fashion-MNIST dataset for this project and we create a model by data partitioning, training using gradient descent, we tune hyper-parameters so that we achieve better performance on validation set. After all this we fix the hyperparameters and test our model's performance on the testing set. This will tell us the effective power gained by the learning.

1 Introduction

The basic idea behind a neural network is to simulate (copy in a simplified but reasonably faithful way) lots of densely interconnected brain cells inside a computer so you can get it to learn things, recognize patterns, and make decisions in a humanlike way. The amazing thing about a neural network is that you don't have to program it to learn explicitly: it learns all by itself, just like a brain! It consists of thousands of neurons which can be categorized into different units like input, hidden and output units. Basically, Neurons are categorized into different layers and neurons from one layer are connected to the other layer and are called weights.

When it's learning or operating normally, patterns of information are fed into the network via the input units, which trigger the layers of hidden units, and these in turn arrive at the output units. This common design is called a feedforward network. For a neural network to learn, there has to be an element of feedback involved which will enhance or improve its learning behavior and output prediction, this process is called backpropagation. Once the network has been trained with enough learning examples, we feed it with an entirely new set of inputs it's never seen before and see how it responds to measure the model's accuracy. The basic structure of a neural network is given below.



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.

2 Dataset:

For training and testing of our classifiers, we are using 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, and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image. Each training and test example is assigned to one of the labels as shown in table below.

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

3 Pre – Processing:

Part1, Part2, Part3:

- **Data Splitting:**

  The Fashion MNIST dataset is originally partitioned into a training set

  and a testing set.

- **Normalization:**

  As it is an image data we simply scale by 1/255 so that pixel intensity range is bound to 0 and 1

- **One Hot Vector (Encoding):**

  One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

  keras.utils.to_categorical

4 Architecture

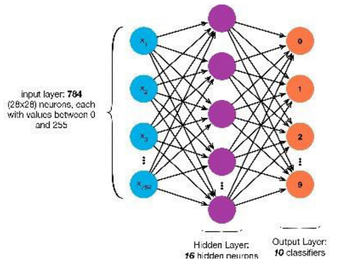
Overview:

We are using Fashion MNIST data set to create a neural model by data partitioning, training using gradient descent, we tune hyper-parameters so that we achieve better performance on validation set and test its performance in Part1.

In Part2 we build a multilayer neural model using keras library and perform the same as of part1 and in part3 we build a convolutional neural model using the keras library with a group of hyperparameters and perform the same as of part1.

Part1:

Computational Graph:


input layer: 784 (28x28) neurons, each with values between 0 and 255
x_1
x_2
x_3
x_{784}
0
1
2
9
Hidden Layer: 16 hidden neurons
Output Layer: 10 classifiers

Forward Propagation:

Information flows through a neural network in two ways. When it's learning (being trained) or operating normally (after being trained), patterns of information are fed into the network via the input units, which trigger the layers of hidden units, and these in turn arrive at the output units. This common design is called a **feedforward network**. Not all unit's "fire" all the time. Each unit receives inputs from the units to its left, and the inputs are multiplied by the weights of the connections they travel along. Every unit adds up all the inputs it receives in this way and if the sum is more than a certain threshold value, the unit "fires" and triggers the units it's connected to (those on its right).

In the implementation we have used sigmoid as an activation function for hidden layer and at output layer we have used SoftMax as an activation function. We use cross entropy to calculate the loss.

$$Z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \text{softmax}(z^{[2]})$$
$$L(a^{[2]}, y) = -\sum y \log a^{[2]}$$

Backward Propagation:

For a neural network to learn, there has to be an element of feedback involved - which is given by a feedback process called **backpropagation**. This involves comparing the output a network produces with the output it was meant to produce, and using the difference between them to modify the weights of the connections between the units in the network, working from the output units through the hidden units to the input units—going backward, in other words. In time, backpropagation causes the network to learn, reducing the difference between actual and intended output to the point where the two exactly coincide, so the network figures things out exactly as it should.

Equations for Backward propagation:

$$\Delta a^{[2]} = \frac{\partial L}{\partial a^{[2]}}$$

$$\Delta z^{[2]} = \frac{\partial L}{\partial z^{[2]}} = a^{[2]} - y$$

$$\Delta a^{[1]} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}}$$

$$\underline{\Delta a^{[1]} = (a^{[2]} - y) W^{[2]}}$$

$$\Delta z^{[1]} = \Delta a^{[1]} \frac{\partial a^{[1]}}{\partial z^{[1]}}$$

$$\Delta W^{[2]} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}}$$

$$\boxed{\Delta W^{[2]} = (a^{[2]} - y) a^{[1]}}$$

$$\Delta z^{[1]} = (a^{[2]} - y) W^{[2]} a^{[1]} (1 - a^{[1]})$$

$$\Delta W^{[1]} = \Delta z^{[1]} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

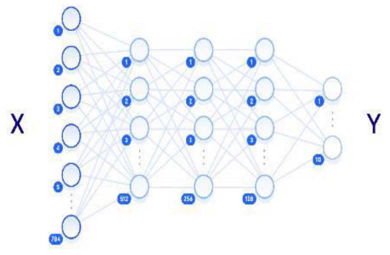
$$\boxed{\Delta W^{[1]} = (a^{[2]} - y) W^{[2]} a^{[1]} (1 - a^{[1]}) x}$$

The goal is to minimize the loss by means of increasing or decreasing the weights, which is commonly called fitting. Which weights should be bigger and which should be smaller can be decided by a function called Gradient descent. The Gradient descent is just the derivative of the loss function with respect to its weights. After the gradient is calculated the weights are updated.

We then validate our data with the updated theta(weights) and bias and calculate the validation loss.
We also test our data with the updated theta(weights) and bias and measure the accuracy of the model on the output of that data.

Part2

Computational Graph:

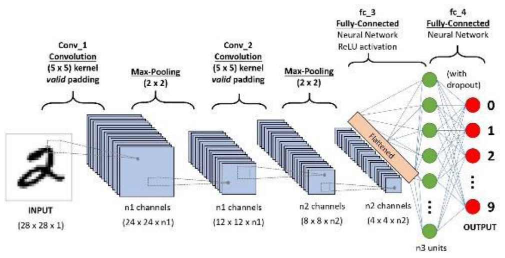


Even multilayer uses the same architecture as that of single layer neural network. But the number of hidden layers is more than one and we can use different activation functions for these layers.
The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero.
Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

We have used keras library to build the model. First three hidden layer that is implemented is a dense hidden layer with the activation function of Rectified Linear Unit (ReLU) and the number of hidden nodes are 200, 300 and 50 respectively. The input is flattened before providing it to the input layer. We have used SoftMax as the activation function for the output layer.

Part3

Computational Graph:



A Convolutional Neural Network 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 role of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

The input image is reshaped and convolved with a kernel of size 2 and 64 filters and then it is maxpooled. 20% of the data is dropped and then sent to the next convoluted layer which has 32 filters, kernel size of 2 and it is maxpooled. We then flatten the data and pass it to layer with relu activation. We use SoftMax as the activation function in the output layer

5 Results:

The task to be performed is:

1. Plot graph showing below details for all 3 classifiers (Neural Network with single hidden layer, multi-layer neural network and convolutional neural network):
   I. accuracy vs number of epochs
   II. loss vs number of epochs
2. Calculate accuracy and construct a confusion matrix for 3 classifiers.

1) Graphs:

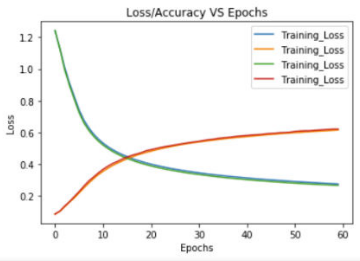


Figure1: Task1 Loss/Accuracy vs Epochs (Train and Validation)

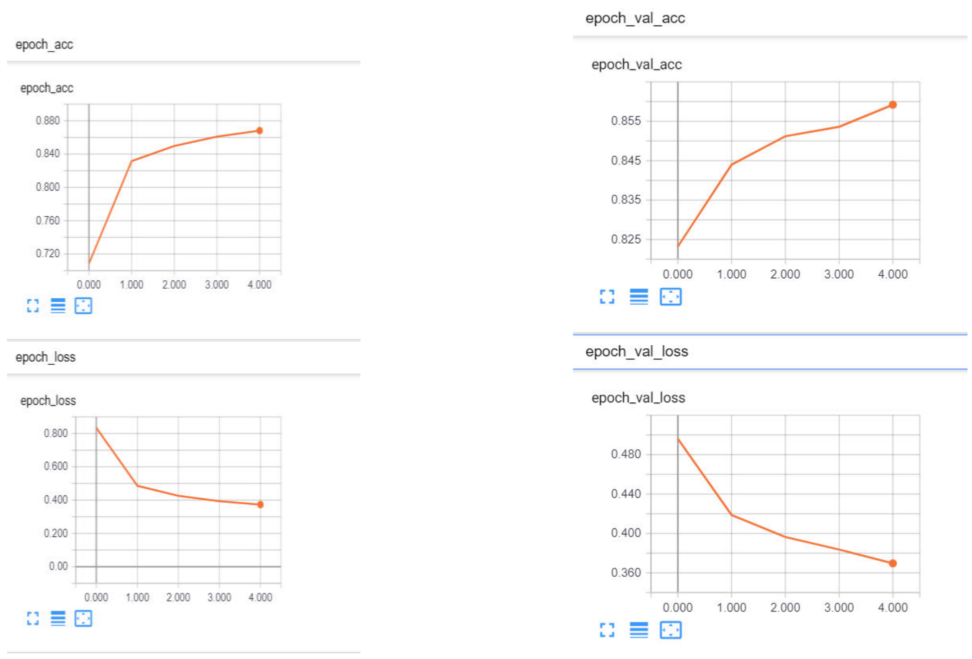


Figure2: Task2 Loss/Accuracy vs Epochs (Train and Validation)

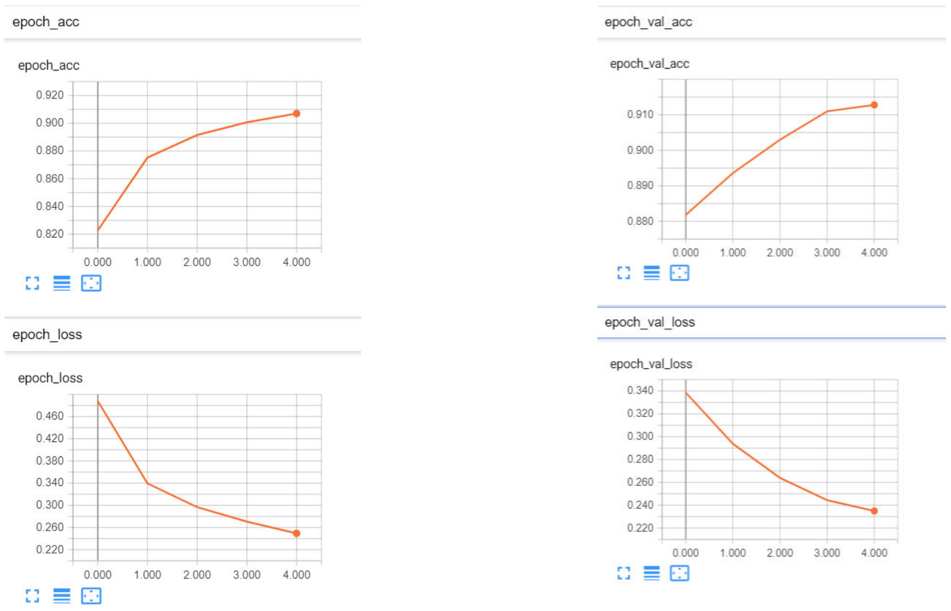
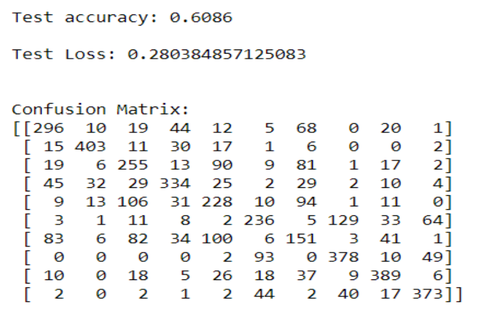


Figure3: Task3 Loss/Accuracy vs Epochs (Train and Validation)

2) **Confusion Matrix and Accuracy:**

Confusion Matrix is nothing but a tabular representation of Actual vs Predicted values. This helps us to find the accuracy of the model and avoid over-fitting.

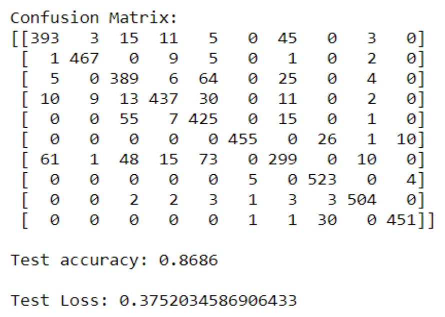


```
Test accuracy: 0.6086

Test Loss: 0.280384857125083


Confusion Matrix:
[[296  10  19  44  12   5  68   0  20   1]
 [ 15 403  11  30  17   1   6   0   0   2]
 [ 19   6 255  13  90   9  81   1  17   2]
 [ 45  32  29 334  25   2  29   2  10   4]
 [  9  13 106  31 228  10  94   1  11   0]
 [  3   1  11   8   2 236   5 129  33  64]
 [ 83   6  82  34 100   6 151   3  41   1]
 [  0   0   0   0   2  93   0 378  10  49]
 [ 10   0  18   5  26  18  37   9 389   6]
 [  2   0   2   1   2  44   2  40  17 373]]
```

Figure4: Task1 Confusion Matrix and Accuracy

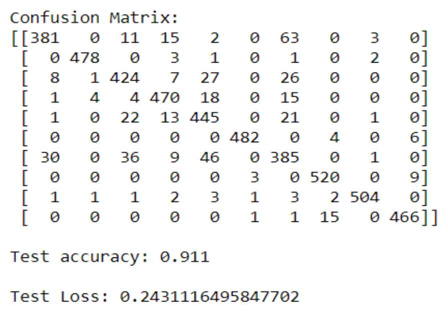


```
Confusion Matrix:
[[393   3  15  11   5   0  45   0   3   0]
 [  1 467   0   9   5   0   1   0   2   0]
 [  5   0 389   6  64   0  25   0   4   0]
 [ 10   9  13 437  30   0  11   0   2   0]
 [  0   0  55   7 425   0  15   0   1   0]
 [  0   0   0   0   0 455   0  26   1  10]
 [ 61   1  48  15  73   0 299   0  10   0]
 [  0   0   0   0   0   5   0 523   0   4]
 [  0   0   2   2   3   1   3   3 504   0]
 [  0   0   0   0   0   1   1  30   0 451]]

Test accuracy: 0.8686

Test Loss: 0.3752034586906433
```

Figure5: Task2 Confusion Matrix and Accuracy



```
Confusion Matrix:
[[381   0  11  15   2   0  63   0   3   0]
 [  0 478   0   3   1   0   1   0   2   0]
 [  8   1 424   7  27   0  26   0   0   0]
 [  1   4   4 470  18   0  15   0   0   0]
 [  1   0  22  13 445   0  21   0   1   0]
 [  0   0   0   0   0 482   0   4   0   6]
 [ 30   0  36   9  46   0 385   0   1   0]
 [  0   0   0   0   0   3   0 520   0   9]
 [  1   1   1   2   3   1   3   2 504   0]
 [  0   0   0   0   0   1   1  15   0 466]]

Test accuracy: 0.911

Test Loss: 0.2431116495847702
```

Figure6: Task3 Confusion Matrix and Accuracy

From the confusion matrix we can observe that the number of accurate predictions increases from task1 to task2 to task3. Task3 has highest number of accurate predictions because of which its accuracy is the best, followed by Task2 and the least is Task1.

Using trial and error method we input different values for hyperparameter like epochs and learning rate, hidden nodes, number of layers and observe the loss and other parameters, to judge the best hyperparameters for the model. After tuning all the hyperparameters we fix the hyperparameters and test the model and the results are as below:

	Accuracy	**Loss**
Task1	60.86%	28.03%
Task2	86.86%	37.5%
Task3	91.1%	24.3%

6 Conclusion

For the given problem statement, we used neural network with single hidden layer, neural network with multiple hidden layer and convolutional neural network. For the given data we normalized, split the data, trained and validated it using forward and backward propagation. Later we used the obtained values of weights and bias from the trained data to perform the test to measure the accuracy. We also performed different trials for different hyperparameters like epoch and learning rate and chose the best fit for our model. Among the three different models, we can observe that multi-layer neural network performs better than single layer neural network because multi-layer means more non-linearities applied to the data. So, among the three tasks, CNN performs the best using the optimal hyperparameters.

7 Acknowledgements

We are extremely grateful to Professor Sargur Srihari for elucidating all the necessary concepts related to Machine Learning. We also thank all the TAs for their efforts taken in clearing our doubts/questions.

8 References

[1] *Pattern Recognition and Machine Learning (PRML) by Chris Bishop (Springer 2006)*

[2] *https://www.explainthatstuff.com/introduction-to-neural-networks.html*

[3] *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*

[4] *http://cs231n.github.io/convolutional-networks/*

[5] *https://www.coursera.org/learn/introduction-tensorflow/home/week/3*

[6] *https://www.coursera.org/learn/convolutional-neural-networks/home/week/1*

[7] *https://www.coursera.org/learn/neural-networks-deep-learning/home/week/4*

[8] *Recitation notes [Equations2.pdf]*

[9] *https://www.tensorflow.org/tutorials/keras/classification*

[10] *https://www.tensorflow.org/tensorboard/scalars_and_keras*