Classification Using Neural Network and Convolutional Neural Network

Sanidhya Chopde
Department of Computer Science
University at Buffalo
Buffalo, NY 14260
schopde@buffalo.edu

Abstract

The aim is to implement neural network and convolutional neural network for the task of classification of images from the Fashion-MNIST dataset into one of ten classes.

1 Introduction

In this project the task is to perform classification using implementation of neural network with one hidden layer, a multi-layer neural network and a convolutional neural network. For the training process the Fashion-MNIST clothing images will be used. We will use this dataset to classify the images into one of the ten different classes.

2 Dataset Definition

For this project we will be using the Fashion-MNIST dataset for training, testing and validation. The dataset is of Zalando's article images consisting of a training set of 60000 examples and a testing set of 10000 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 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 1.

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

Table 1. for Fashion-MNIST dataset

3 Pre-Processing

3.1 Neural network with one hidden layer

The preprocessing phase of part 1 involved doing the following things :

- We have to normalize the data by dividing every single example of the training and testing image data with 255. Since every single pixel is of a value between 0 & 255, this step is done so as to get the values of the examples between 1 and 0 which makes it easy to classify.
- Then, we reshape the training labels and test labels to make them compatible for further computations.

3.2 Multi-Layer Neural network

- We have to reshape the input training and testing image data to make it compatible for further computations as keras requires the third dimension.
- We then have to normalize the data by dividing every single example of the training and testing image data with 255. Since every single pixel is of a value between 0 & 255, this step is done so as to get the values of the examples between 1 and 0 which makes it easy to classify.

3.3 Convolutional Neural Network (CNN)

- We have to reshape the input training and testing image data to make it compatible for further computations with keras.
- We also have to do one hot encoding of training and testing labels so as to convert them into a form which could be provided to the algorithm for computation.
- We then have to normalize the data by dividing every single example of the training and testing image data with 255. Since every single pixel is of a value between 0 & 255, this step is done so as to get the values of the examples between 1 and 0 which makes it easy to classify.

4 Architecture

4.1 Fashion-MNIST dataset

The first step is to download and process the Fashion-MNIST dataset. We have to process it into a numpy array which contains the feature vectors and the labels in.

4.2.1 Neural network with one hidden layer

In the part one of this project, gradient descent for neural networks is used to train the model using a group of hyperparameters. A feedforward neural network has been used here. The feedforward neural network contains multiple neurons(nodes) arranged in layers. Nodes from adjacent layers have connections or edges between them. All these connections have weights associated with them. A feedforward neural network can consist of three types of nodes:

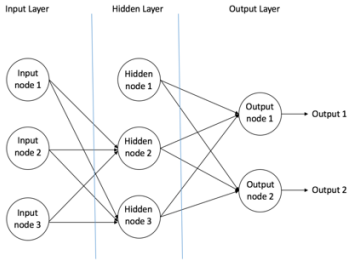

Figure 1: Feedforward Neural Network

1. Input Nodes – The Input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.
2. Hidden Nodes – The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.
3. Output Nodes – The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network [3] (this property of feed forward networks is different from Recurrent Neural Networks in which the connections between the nodes form a cycle).

4.2.2 Sigmoid function

In order to map predicted values to probabilities, we use the sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

$$S(z) = \frac{1}{1 + e^{-z}}$$

4.2.3 Gradient Descent

To reduce our cost, we use gradient descent. Gradient descent comprises two steps:

- calculating gradients of the loss/error function,
- Updating existing parameters in response to the gradients, which is how the descent is done.

This cycle is repeated until reaching the minima of the loss function.

4.3.1 Multi-Layer Neural network

A multi-layer neural network functions in the same way as a single layer neural network that we saw above with the only difference being that we add multiple hidden layers instead of one. Adding more hidden layers means more neurons which helps the model to get trained in a better way resulting in increased accuracy. For creating a multi-layer neural network, the Keras library has been used.

4.3.2 Keras Library and creating a model

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Keras is a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

In this project, sequential class has been used to build the model.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(256, activation='sigmoid'),
    keras.layers.Dense(150, activation='sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])
```

Figure 2: Code for building model using sequential class

The first layer in this network, keras.layers.Flatten, transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels).

After the pixels are flattened, the network consists of a sequence of three keras.layers.Dense layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 128 nodes (or neurons), the second one has 150 nodes and the last layer is a 10-node *softmax* layer that returns an array of 10 probability scores that sum to 1. Each node contains a score that indicates the probability that the current image belongs to one of the 10 classes.

Before the model is ready for training, it needs a few more settings. These are added during the model's compile step:

- Loss function: This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.
- Optimizer: This is how the model is updated based on the data it sees and its loss function.
- Metrics: Used to monitor the training and testing steps.

Training the neural network model requires the following steps:

- Feed the training data to the model.
- The model learns to associate images and labels.
- The model is asked to make predictions about a test set.

After training the test accuracy is evaluated for the trained model.

4.4.1 Convolutional Neural Network (CNN)

A multi-layer neural network is usually a fully connected network. This means that each neuron in one layer is connected to all neurons in the next layer. This usually leads to overfitting of data which means that the model is very precisely corresponding to a particular dataset and it may fail to correspond to any new data that is provided to it. Therefor CNN's take a different approach and they assemble ore complex patterns using smaller and simpler patterns. This also means that they are low on scale on being a fully connected network.

4.4.2 Keras Library and creating a CNN model

In this project, a sequential class has been used to build the CNN model too. The Sequential constructor takes an array of Keras Layers. Here 3 types of layers have been used for our CNN: Convolutional2D, Max Pooling, and Dropout.

Before the model is ready for training, we have to perform the same compilation steps to configure the learning process. This is where the loss function, optimizer and the metrics evaluated by model during training and testing is defined

Then the training of the model is performed using the fit () API.

After training the test accuracy is evaluated for the trained model.

5 Result

5.1 Neural network with single hidden layer

The graph of training loss vs number of epochs for the Neural network with single hidden layer classifier with the following hyperparameters:

- No of epochs: 1000
- Learning rate: 2
- Nodes: 70

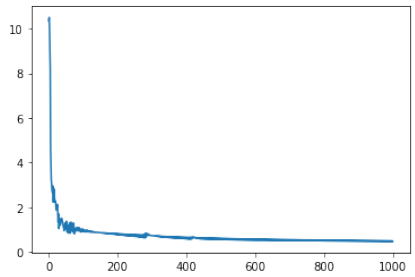

Figure 3: Training loss vs number of epochs

The testing accuracy for this classifier is: 82%

The confusion matrix for this classifier is:

```
[[878   6  28  68   2   1 277   0   7   1]
 [  1 940   2  14   1   0   2   0   0   0]
 [ 34  14 803  20 161   0 210   0  22   0]
 [ 44  37  12 834  34   0  46   0   8   0]
 [ 10   1 128  47 776   1 192   0  11   0]
 [  1   0   0   1   0 902   0  33   7  23]
 [ 10   0  16   5  17   0 246   0   3   0]
 [  0   0   0   0   0  50   0 908   5  37]
 [ 22   2  11  10   9  10  27   3 934   1]
 [  0   0   0   1   0  36   0  56   3 938]]
```

Figure 4: Confusion matrix for Neural network with single hidden layer

5.2 Multi-Layer Neural network

The graph of training loss vs number of epochs for the Neural network with single hidden layer classifier with the following hyperparameters:

- No of epochs: 15
- Number of nodes in first hidden layer: 256
- Number of nodes in second hidden layer: 150
- Number of nodes in first softmax layer: 10

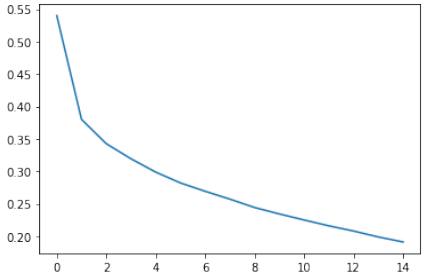

Figure 5: Training loss vs number of epochs

The testing accuracy for this classifier is: 89.02%

The confusion matrix for this classifier is:

```
[[861   2   9  16   5   2 101   0   4   0]
 [  2 972   1  18   3   0   4   0   0   0]
 [ 14   0 756  13 111   0 105   0   1   0]
 [ 29   4  10 882  46   1  24   0   4   0]
 [  1   0  63  14 837   0  84   0   1   0]
 [  0   0   0   1   0 954   0  20   1  24]
 [126   0  51  23  57   0 734   0   9   0]
 [  0   0   0   0   0  11   0 972   0  17]
 [  4   0   2   2   6   2   4   3 977   0]
 [  0   0   0   0   0   4   1  38   0 957]]
```

Figure 6: Confusion matrix for Multi-Layer Neural network

5.3 Convolutional Neural Network

The graph of training loss vs number of epochs for the Convolutional Neural network classifier with the following hyperparameters:

- No of epochs: 10
- Batch size: 70

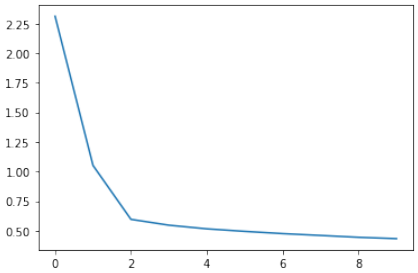

Figure 7: Training loss vs number of epochs

The testing accuracy for this classifier is: 85.44%
The confusion matrix for this classifier is:

```
[[829   5  16  51   4   1  79   0  15   0]
 [  2 961   2  25   5   0   4   0   1   0]
 [ 16   1 730  11 166   0  66   0  10   0]
 [ 28  20  17 880  29   0  21   0   5   0]
 [  1   2  95  51 776   0  72   0   3   0]
 [  0   0   0   1   0 950   0  34   1  14]
 [173   1 107  42  95   0 553   0  29   0]
 [  0   0   0   0   0  20   0 934   0  46]
 [  2   2   5   5   4   1   1   4 975   1]
 [  0   0   0   0   0   5   0  38   1 956]]
```

Figure 8: Confusion matrix for Convolutional Neural network

6 Conclusion

In this project classification was performed using three different classifiers namely neural network with one hidden layer, multi-layer neural network and convolutional neural network. We calculated accuracy and confusion matrix for all three classifiers. The different classifiers gave us different results according the hyperparameters provided to them. At the end we were successfully able to train the model in three different ways to classify the given images to one of the ten classes with considerable accuracy.

Acknowledgements

I am extremely grateful to Professor Sargur Srihari for teaching all the necessary concepts related to logistic regression. I would also like to thank all the TA's of the course for helping me at every step during the course of this project.

References

[1] Professor Sargur Srihari's lecture slides.
[2] Keras for CNN (https://victorzhou.com/blog/keras-cnn-tutorial/)
[3] Fashion-MNIST with tf.keras (https://medium.com/tensorflow/hello-deep-learning-fashion-mnist-with-keras-50fcff8cd74a)
[4] Keras Documentation(https://keras.io/)
[5]Basic classification (https://www.tensorflow.org/tutorials/keras/classification)
[6] Sklearns (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)
[7] Introduction to Neural networks (https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/)
[8] Implementation of gradient descent (https://towardsdatascience.com/a-step-by-step-implementation-of-gradient-descent-and-backpropagation-d58bda486110)
[9] Wikipedia CNN(https://en.wikipedia.org/wiki/Convolutional_neural_network)