# Neural Network Project

**Shageenth Sandrakumar**
Department of Data Science Engineering University
Buffalo, NY 14260
*shageent@buffalo.edu*

**Abstract**

In this lab, we built three projects that study how Neural Network learn. Each project was built on the MNSIT's Fashion Data Set. The first model we built was a simple Neural Network from scratch using python, the second model we built used Keras and Tensor flow to build multiple layer Neral Networks, and the last model we have to build a Convolutional Nerual Network using the Keras module.

## 1 Introduction to Neural Networks

In this section I will spend time talking a basic idea of a Neural Network. Then show a picture of a simple

Neural Network and explain it's components.

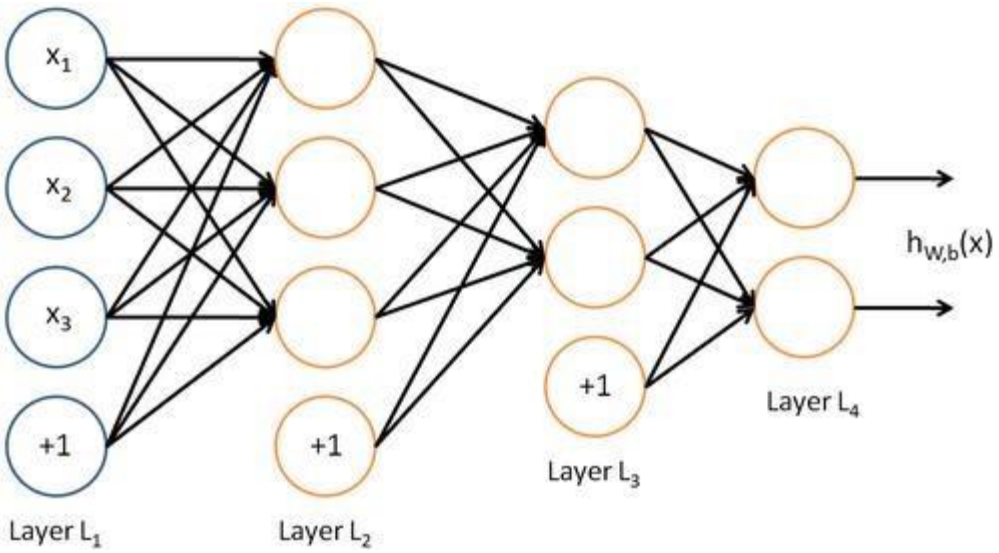
### 1.1 What is a Neural Network?

A Neural Network is actually a function of many variables: It takes an input, makes computations and produces an output. We like to visualize it as neurons in different layers, with each neuron in a layer connected with all neurons in the previous and the next layer. All the computations take place inside those neurons and depend on the weights that connect the neurons with each other. So all we have to do is learn the right weights in order to get the desired output.

Their structure is generally very complex including a lot of layers and even more than a million neurons in order to be able to handle the large datasets of our era. However, in order to understand how large deep neural networks work one should start with the simplest ones.

### 1.2 Example of a Simple Neural Network

Here's an example of a Neural Network:




Neural networks can usually be read from left to right. Here, the first layer is the layer in which inputs are entered. There are 2 internals layers (called hidden layers) that do some math, and one last layer that contains all the possible outputs. Don't bother with the "+1"s at the bottom of every columns. It is something called "bias" and we'll talk about that later.

## 2 Performance of each Neural Network

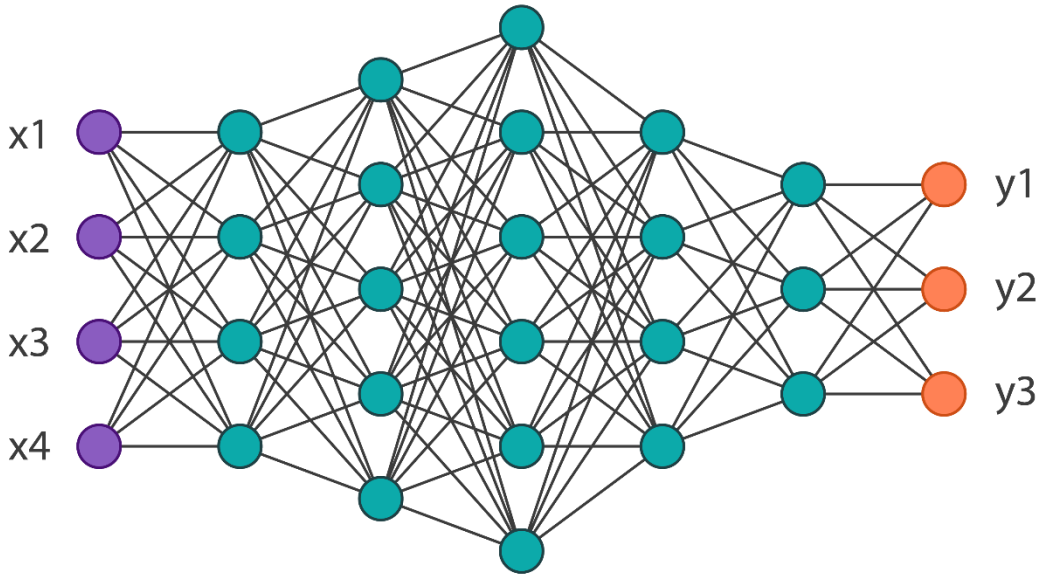
This section will try to explain why different Neural Network might have different performance base on the activation functions and algorithm used.

### 3.1 One Hidden Layer Neural Network

This is the simplest form of a Neural Network. This is where we only have three layers. So we will have an input layer, a hidden layer and an output layer. Next, we need an activation function. The sigmoid function is a good choice for the last layer because it outputs values between 0 and 1 while tanh (hyperbolic tangent) was considered but the sigmoid had performed better for the hidden layer, but every other commonly used function will work (RELU etc.)

### 3.2 Mutli-Layer Model Neural Network

This is a larger version of the last Neural Network with multiple weights and hidden layers. Since Keras is easier to use more complicated activation functions were used such as Hyperbolic Tangent and Softmax function. These were used to make the results more accurate. Here is a picture of a larger NN:

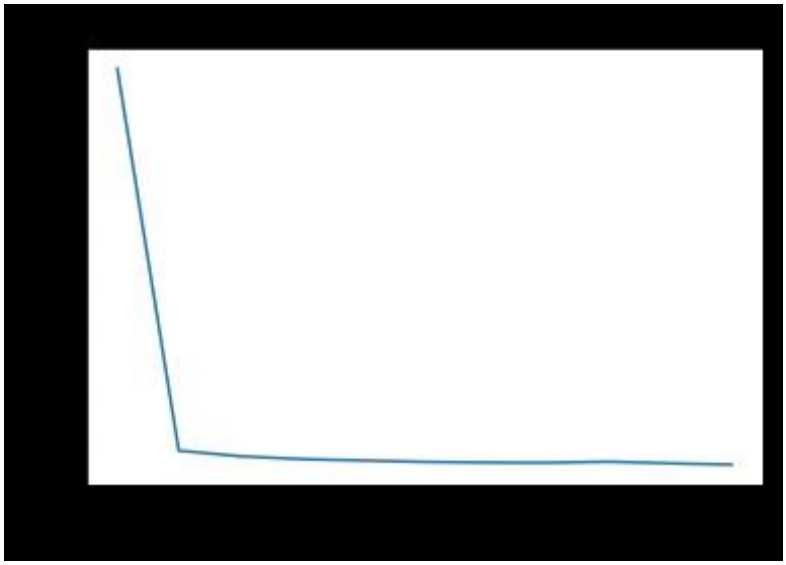



### 3.3 Convolution Neural Network

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 preprocessing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

## 4 Result Comparison

### 1. One hidden layer Neural Network



The Cost rapidly declines and then steadily declines after the 100th iteration. However, our model isn't quite precise:

```
  after removing the cwd from sys.path.
1]: 10000

2]: accuracy = np.sum(np.multiply(y_predict,Y_test))/len(Y_test[0])*100
    print(accuracy)

    69.43
```

### 2. Nine Layer Model Neural Network

```
y_train_pred = model.predict_classes(X_train_centered, verbose=0)
print('First 3 predictions: ', y_train_pred[:3])
```

```
First 3 predictions:  [9 0 0]
```

```
y_train_pred = model.predict_classes(X_train_centered, verbose=0)
correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]

print(f'Training accuracy: {(train_acc * 100):.2f}')

# calculate testing accuracy
y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]

print(f'Test accuracy: {(test_acc * 100):.2f}')
```

```
Training accuracy: 92.77
Test accuracy: 87.36
```

This came out to be fairly accurate. 87% and was an overall good model.

### 3. Convolutional Neural Network

```
Epoch 3/3
60000/60000 [==============================] - 129s 2ms/step - loss: 0.2788 - acc: 0.8976 - val_loss: 0.3638 - val
_acc: 0.8718
```

```
<keras.callbacks.History at 0x7f34da40fe80>
```

```
#predict first 4 images in the test set
y_dummy = model.predict(X_test[:4])
y_dummy = np.round(y_dummy)
print(y_dummy)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
np.sum(y_dummy - y_test[:4])
```

```
0.0
```

```
#PERFECT MATCHING!!!!! AWESOME
```

It faired up around the same as the MultiLayer Network

## CONCLUSION

The Convolution Neural Network are the most accurate out of the three, because it involves performing a convolution algorithm which essentially recognize the image and extract 119 abstract features from the image. It could achieve 90% accuracy when train well enough. The second model is the multiple layer Neural network which has 87% accuracy, it shows that the more layer we had, the better prediction outcome. The first layer has around 60% accuracy after train, this is due to not having a large number of iterations (due to time constraints)