Report: Assignment 1
COMP 8740: Neural Networks
ID: U00744746

Hosneara Ahmed

September 14, 2021

1 Introduction

Deep Neural Network (DNN) has been popular recently for solving complex machine learning problems. Since we are living in a big data era, day by day data size is becoming larger which is quite difficult for a general machine learning algorithm to deal with. However, in case of deep neural networks, with varying number of layers it can extract complex patterns as the amount of data grows. In this assignment, we have explored two different DNN with varying number of layers as well as depth of the network and observed the prediction accuracy and computational times.

1.1 Dataset

MNIST dataset contains 70,000 image examples each of size 28x28 pixels. It is already in keras package under tensorflow. Each image is a digit from 0 to 9 inclusive, so there are 10 image labels. These images are grayscale which means there is only one channel in contrast to 3-channel RGB colorful images. So each pixel can take 0 to 255 values. Given an image, the target is to predict what number it represents among the 10 distinct digit classes.

2 Methodology

Suppose, images are $X = \{x_1, x_2, x_n\} x_i \in R^{dxd}$. Each image is a grayscale image which means there is one channel ranging values from 0 to 255. Each image represents a digit from 0 to 9. For example, in an image of digit 9 will have darker color (close to 0) only in those pixels such that is looks like a '9'. Lighter color (close to 255) in other pixels except the pixels where the digit is.

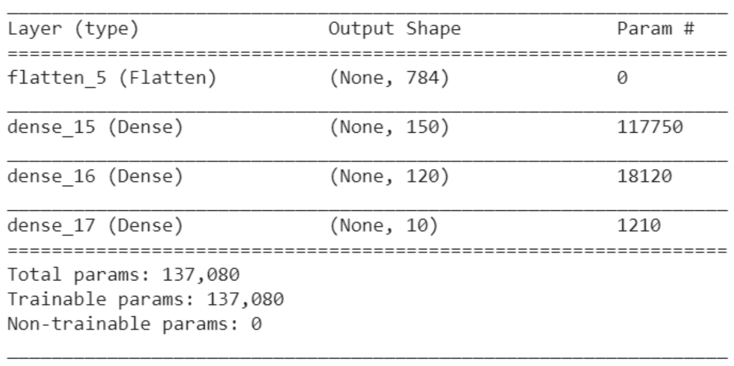
3 Deep Learning Architecture

There are two configurations for designing the DNN.

- $784 \rightarrow 150 \rightarrow 120 \rightarrow 10$: There is one input layer with 784 neurons. This comes from flattening the image of 28x28 pixel size. Then there are 2 hidden layers with 150 and 120 neurons respectively followed by the output layer. In the output layer there are 10 neurons to classify the 10 digits.

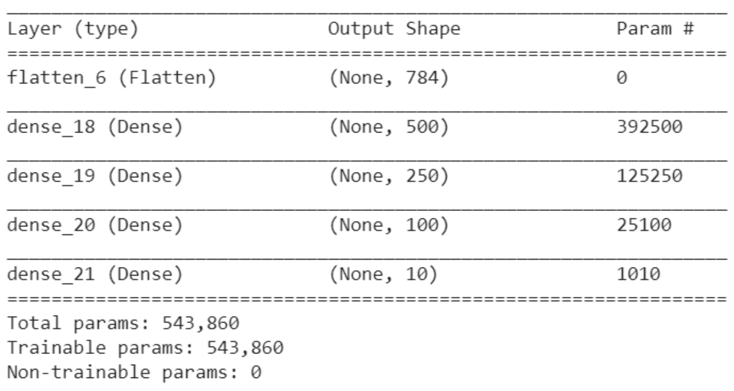
- $784 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 10$: This configuration is similar to the first one except in the number of hidden layers as well as number of neurons in each layer. Here it has 3 hidden layers which contains 500, 250, 100 neurons respectively.

The base model is sequential which means it stacks a sequence of layers to the model. Following that, the first layer is flattening the inputs to a vector. Then in each hidden layer and the output layer is Dense layer which means



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_5 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_15 (Dense)             (None, 150)               117750
_________________________________________________________________
dense_16 (Dense)             (None, 120)               18120
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1210
=================================================================
Total params: 137,080
Trainable params: 137,080
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Model Architecture: DNN config 784 $\rightarrow$ 150 $\rightarrow$ 120 $\rightarrow$ 10



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_6 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_18 (Dense)             (None, 500)               392500
_________________________________________________________________
dense_19 (Dense)             (None, 250)               125250
_________________________________________________________________
dense_20 (Dense)             (None, 100)               25100
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1010
=================================================================
Total params: 543,860
Trainable params: 543,860
Non-trainable params: 0
```

Figure 2: Model Architecture: DNN config 784 $\rightarrow$ 500 $\rightarrow$ 250 $\rightarrow$ 100 $\rightarrow$ 10

all the neurons are connected to each other. The hidden layer applies sigmoid activation and the output layer applies softmax activation. Sigmoid activation converts the output of a forward propagation layer between 0 and 1. And in the softmax layer the output is a probability distribution for the classes. So the classified class will have highest probability across all the classes. And the probabilities will sum up to one.

Suppose the number of neurons at any layer l is n_l. Now number of parameter at layer l with bias is calculated as-

$$n_{l-1} * (n_l + 1)$$

For example, in the first Dense layer of the configuration-1 model has parameters $150 * (784 + 1) = 117750$

4 Experiment and Results

I have done the experiment in Google Colab. At first the MNIST label dataset is encoded to one hot tensor. This means each vector in the tensor has 1 in the corresponding position of the digit that the image represents. It is required because we have used categorical crossentropy that means we have considered this as a categorical problem. Optimizer is Stochastic Gradient Descent, it is an iterative method that update the parameter after each training point. Loss is categorical crossentropy because the last layer activation function was softmax, so it is a multi-class classification problem. The metric for evaluation is selected as accuracy. Mini batch size is 64 as instructed. It fit in the RAM in the Google Colab and did not give any memory overload problem. I have trained the model in 50 epochs and observed the loss if it is gradually decreasing.

4.1 Training Logs

- **Training logs for configuration** $784 \to 150 \to 120 \to 10$

```
Epoch 1/50
938/938 [==============================] - 3s
    3ms/step - loss: 2.2557 - accuracy: 0.2484
Epoch 2/50
938/938 [==============================] - 2s
    2ms/step - loss: 2.0961 - accuracy: 0.4778
Epoch 3/50
938/938 [==============================] - 2s
    2ms/step - loss: 1.8043 - accuracy: 0.6016
Epoch 4/50
938/938 [==============================] - 2s
    2ms/step - loss: 1.4156 - accuracy: 0.6860
Epoch 5/50
938/938 [==============================] - 2s
    3ms/step - loss: 1.0989 - accuracy: 0.7561
Epoch 6/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.8961 - accuracy: 0.7937
Epoch 7/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.7622 - accuracy: 0.8156
Epoch 8/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.6698 - accuracy: 0.8298
Epoch 9/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.6040 - accuracy: 0.8431
Epoch 10/50
```

```
938/938 [==============================] - 2s
    2ms/step - loss: 0.5552 - accuracy: 0.8528
Epoch 11/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.5173 - accuracy: 0.8627
Epoch 12/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.4866 - accuracy: 0.8691
Epoch 13/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.4614 - accuracy: 0.8766
Epoch 14/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.4403 - accuracy: 0.8812
Epoch 15/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.4225 - accuracy: 0.8855
Epoch 16/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.4073 - accuracy: 0.8894
Epoch 17/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3943 - accuracy: 0.8926
Epoch 18/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3833 - accuracy: 0.8949
Epoch 19/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3736 - accuracy: 0.8964
Epoch 20/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3651 - accuracy: 0.8982
Epoch 21/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3576 - accuracy: 0.8998
Epoch 22/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3509 - accuracy: 0.9007
Epoch 23/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3449 - accuracy: 0.9022
Epoch 24/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3394 - accuracy: 0.9032
Epoch 25/50
```

```
938/938 [==============================] - 2s
    2ms/step - loss: 0.3343 - accuracy: 0.9045
Epoch 26/50
938/938 [==============================] - 2s
    2ms/step - loss: 0.3298 - accuracy: 0.9055
Epoch 27/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3253 - accuracy: 0.9068
Epoch 28/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3214 - accuracy: 0.9080
Epoch 29/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3175 - accuracy: 0.9090
Epoch 30/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3139 - accuracy: 0.9097
Epoch 31/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3104 - accuracy: 0.9105
Epoch 32/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3073 - accuracy: 0.9112
Epoch 33/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3041 - accuracy: 0.9125
Epoch 34/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.3010 - accuracy: 0.9130
Epoch 35/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.2982 - accuracy: 0.9136
Epoch 36/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.2953 - accuracy: 0.9146
Epoch 37/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.2927 - accuracy: 0.9156
Epoch 38/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.2900 - accuracy: 0.9154
Epoch 39/50
938/938 [==============================] - 2s
    3ms/step - loss: 0.2874 - accuracy: 0.9167
Epoch 40/50
```

```
938/938 [==============================] - 2s
   3ms/step - loss: 0.2850 - accuracy: 0.9171
Epoch 41/50
938/938 [==============================] - 2s
   2ms/step - loss: 0.2825 - accuracy: 0.9183
Epoch 42/50
938/938 [==============================] - 2s
   2ms/step - loss: 0.2802 - accuracy: 0.9185
Epoch 43/50
938/938 [==============================] - 2s
   3ms/step - loss: 0.2778 - accuracy: 0.9192
Epoch 44/50
938/938 [==============================] - 2s
   3ms/step - loss: 0.2756 - accuracy: 0.9204
Epoch 45/50
938/938 [==============================] - 2s
   3ms/step - loss: 0.2734 - accuracy: 0.9205
Epoch 46/50
938/938 [==============================] - 2s
   3ms/step - loss: 0.2712 - accuracy: 0.9209
Epoch 47/50
938/938 [==============================] - 2s
   2ms/step - loss: 0.2690 - accuracy: 0.9219
Epoch 48/50
938/938 [==============================] - 2s
   2ms/step - loss: 0.2668 - accuracy: 0.9226
Epoch 49/50
938/938 [==============================] - 2s
   3ms/step - loss: 0.2648 - accuracy: 0.9229
Epoch 50/50
938/938 [==============================] - 2s
   3ms/step - loss: 0.2627 - accuracy: 0.9237
```

- **Training logs for configuration** $784 \to 500 \to 250 \to 100 \to 10$

```
Epoch 1/50
938/938 [==============================] - 6s
    6ms/step - loss: 2.3043 - accuracy: 0.1221
Epoch 2/50
938/938 [==============================] - 5s
    6ms/step - loss: 2.2819 - accuracy: 0.1673
Epoch 3/50
938/938 [==============================] - 5s
    6ms/step - loss: 2.2629 - accuracy: 0.2302
Epoch 4/50
938/938 [==============================] - 5s
    5ms/step - loss: 2.2321 - accuracy: 0.3145
Epoch 5/50
938/938 [==============================] - 5s
    6ms/step - loss: 2.1717 - accuracy: 0.3896
Epoch 6/50
938/938 [==============================] - 5s
    6ms/step - loss: 2.0420 - accuracy: 0.4506
Epoch 7/50
938/938 [==============================] - 5s
    6ms/step - loss: 1.8173 - accuracy: 0.5224
Epoch 8/50
938/938 [==============================] - 5s
    6ms/step - loss: 1.5471 - accuracy: 0.5967
Epoch 9/50
938/938 [==============================] - 5s
    6ms/step - loss: 1.2690 - accuracy: 0.6560
Epoch 10/50
938/938 [==============================] - 5s
    5ms/step - loss: 1.0580 - accuracy: 0.7018
Epoch 11/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.9292 - accuracy: 0.7264
Epoch 12/50
938/938 [==============================] - 5s
    5ms/step - loss: 0.8474 - accuracy: 0.7474
Epoch 13/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.7875 - accuracy: 0.7655
Epoch 14/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.7378 - accuracy: 0.7818
Epoch 15/50
938/938 [==============================] - 5s
```

```
    6ms/step - loss: 0.6930 - accuracy: 0.7980
Epoch 16/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.6510 - accuracy: 0.8126
Epoch 17/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.6127 - accuracy: 0.8250
Epoch 18/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.5789 - accuracy: 0.8354
Epoch 19/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.5492 - accuracy: 0.8442
Epoch 20/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.5242 - accuracy: 0.8510
Epoch 21/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.5027 - accuracy: 0.8575
Epoch 22/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4840 - accuracy: 0.8628
Epoch 23/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4676 - accuracy: 0.8678
Epoch 24/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4532 - accuracy: 0.8717
Epoch 25/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4405 - accuracy: 0.8757
Epoch 26/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4291 - accuracy: 0.8784
Epoch 27/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4190 - accuracy: 0.8815
Epoch 28/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4100 - accuracy: 0.8842
Epoch 29/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.4017 - accuracy: 0.8857
Epoch 30/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3941 - accuracy: 0.8882
```

```
Epoch 31/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3873 - accuracy: 0.8894
Epoch 32/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3811 - accuracy: 0.8911
Epoch 33/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3753 - accuracy: 0.8928
Epoch 34/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3700 - accuracy: 0.8946
Epoch 35/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3650 - accuracy: 0.8958
Epoch 36/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3606 - accuracy: 0.8965
Epoch 37/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3563 - accuracy: 0.8982
Epoch 38/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3522 - accuracy: 0.8989
Epoch 39/50
938/938 [==============================] - 5s
    5ms/step - loss: 0.3484 - accuracy: 0.9000
Epoch 40/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3450 - accuracy: 0.9011
Epoch 41/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3415 - accuracy: 0.9021
Epoch 42/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3384 - accuracy: 0.9022
Epoch 43/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3354 - accuracy: 0.9035
Epoch 44/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3326 - accuracy: 0.9039
Epoch 45/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3297 - accuracy: 0.9048
Epoch 46/50
```

```
938/938 [==============================] - 5s
    6ms/step - loss: 0.3270 - accuracy: 0.9055
Epoch 47/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3246 - accuracy: 0.9061
Epoch 48/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3221 - accuracy: 0.9072
Epoch 49/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3196 - accuracy: 0.9077
Epoch 50/50
938/938 [==============================] - 5s
    6ms/step - loss: 0.3173 - accuracy: 0.9085
```

4.2 Discussion and Comparison

From both configuration, the accuracy increases with epochs and losses decreases. However, we can see, in the fist case, when the number of hidden layer is smaller, the convergence to accuracy 90% is achieved in 22nd epoch. However, in case of second configuration, when number of hidden layer is more, then it is taking much longer time to converge and gets to 90% accuracy at 39th epoch.

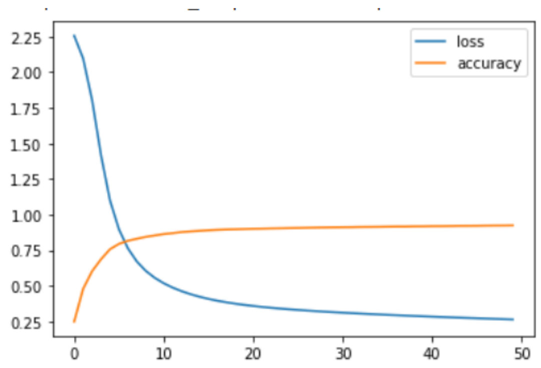


Figure 3: Loss vs Accuracy: DNN config 784 → 150 → 120 → 10

The loss vs accuracy plot for each of the configuration, we see the similar behavior that I have discussed from training logs. We want loss to be lower and accuracy to be higher. So, the point when the loss and accuracy curve are intersecting, that is our goal to achieve. And we want to go to that point faster which is happening when number of hidden layer is smaller. This is because the input data we are dealing with is not that much complex to classify as the images are in grayscale. Additionally, there is no noise in the image e.g. two or more digits in the same image, one is in background, another in foreground.

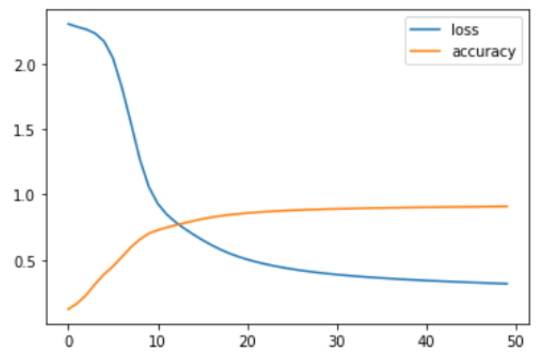


Figure 4: Loss vs Accuracy: DNN config $784 \rightarrow 500 \rightarrow 250 \rightarrow 100 \rightarrow 10$

So, it is pretty simple classification which does not need much deep neural net to solve the problem.

5 Conclusion

We have experimented with one of the hyperparameters that is number of hidden layers in DNN to classify digits in MNIST dataset. Due to simple classification problem with much clean images, a shallower DNN is sufficient and necessary to classify the digits in comparison to deeper DNN to handle complex patterns. So, whenever we have to choose hyperparameters like number of layers in a DNN, we have to choose the one that converges fast and provides better accuracy. If both have same accuracy, then whichever is simpler, choose that one. The reason behind this is, we need to keep the number of parameters small as it is costly to learn the parameters especially in a Dense network where all the nodes are connected to each other.