(Convolution) Artificial Neural Network

Chan Shu You
Department of Computer Science
University at Buffalo
chanshuy@buffalo.edu

Abstract
Artificial neural network helps Identify and categorize fashion products. The data set for this project is Fashion-MNIST data set. 60,000 samples are used for training, 10,000 are used for testing, and 12,000 (20% of the training data) was used for the validation purpose.

1. Introduction
Before neural network was introduced, regression (both linear and logistic) was used to classify objects to two or more different categories. However, the ability the model to classify objects was very limited, and often inaccurate. Thus, Artificial Neural Network was later introduced, which is more complicated than a regular regression and yet more accurate and powerful. The model is inspired by neuron connections in human brain, and thus the neural network model have two or more layers other than the input layer. Since there is no free lunch, neural network model will take longer time to train, but for the trade off the accuracy increases. For this project, we will have 3 different models – Neural Network with 1 hidden layer, Multiple layer neural network, and a convolution neural network. What separates convolution neural network from the rest is that it shares the parameters when training, which reduces the training time. However, the efficiency and the accuracy for CNN model hugely depends on how the architecture is built.

2. Data
The data set we have is Wisconsin Diagnostic Breast Cancer (WDBC) data set. It contains The data set is Fashion-MNIST data set. We will use 60,000 of the sample data for training, 10,000 of the sample data for testing, and 12,000 (20% of the training data) for the validation.

3. Preprocessing

3.1 Modifying Data Set
In order to use the data properly, we need to modify it. For instance, the default data shape is (1, 784), which is the flattened version of image. When the convolution neural network reads data, we need to reshape it to a 3D image shape, which in this case is (28, 28, 1). The other two neural network models use the data as input in its original form.

3.2 Data Distribution

To get the validation data from the training data, the function ‘get_validation_data’ randomly generate index from 0 to 59,999 and get the first 12,000 data from training data using the index. The same data set is used for all three models.

4. Architecture

4.1 Neural Network with 1 hidden layer

4.11 Data

The model is constructed with 784 input nodes, 100 hidden layer nodes (which is a hyper parameter that we can adjust later), and 10 output nodes since there are 10 possible outcome. (784 x 100) and (100 x 10) weights are initialized in the range between -1 and 1, and the biases for each layer is initialized to 0. ‘Sigmoid’ and ‘relu’ will be used for the hidden layer’s activation function, which is also a hyper parameter. The other hyper parameters include learning rate, learning rate descent, epoch ,and batch size.

4.12 Feed Forwarding

We multiply the parameters with the corresponding attributes and sum them all. The result is then used in the activation function. The output of the hidden layer will become the input of the second layer, and the matrix multiplication training process continues till the training reaches to the output layer. For the output layer, the activation function is softmax, which assigns probability for each of the 10 estimated outcomes the model makes.

4.13 Back Propagation

To update our parameters, we compare our predictions with the actual values. In order to do that, we need to convert the target to a matching matrix identical to the prediction. One-hot method is used for the conversion. Throughout the training, the error is monitored since it indicates how well the model is trained and its training direction. To update the second layer’s parameters, we multiply the error with the output of the first layer, which will give us a matrix with a dimension identical to the second layer’s parameter. Then the parameters are updated by subtracting the multiplication of the matrix and the learning rate, another hyper parameters for this model. The parameters from the first layer are updated with similar method. The error from the output layer is back-propagated to the hidden layer by multiplying the error with the second layer’s parameters, and then multiply the result with input layer’s data. It will give us a matrix with a dimension identical to the parameters matrix in the first layer, and then we update the parameters again by subtracting the multiplication of the resulting matrix with learning rate.

4.14 Validation

After epoch times of feed-forwarding and back propagation, to validate the usability and the efficiency of the updated parameters, we use the validation set to predict the values using the updated parameters. And then we calculate the Accuracy.

4.15 Testing All the Hyper-Parameters Combinations.

The following are the hyper parameters that we will test:

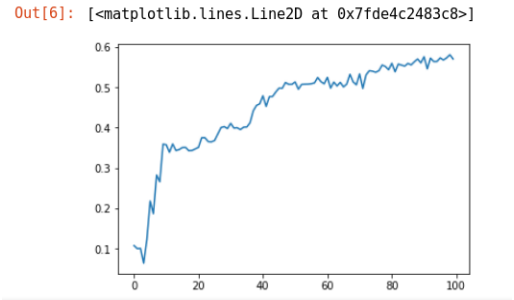
hidden = [[20],[80],[300]]
learning_rate = [0.001, 0.0001, 0.00001]

To find the best hyper parameters, we test out each of them.

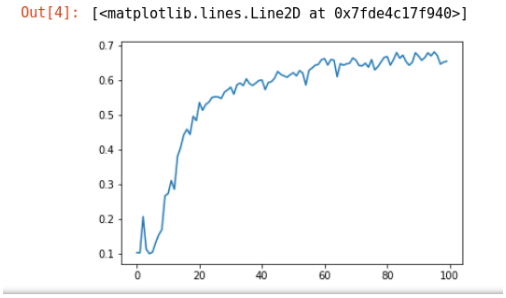
4.16 Result

The following are all the hyper-parameters combinations and the accuracy scores:

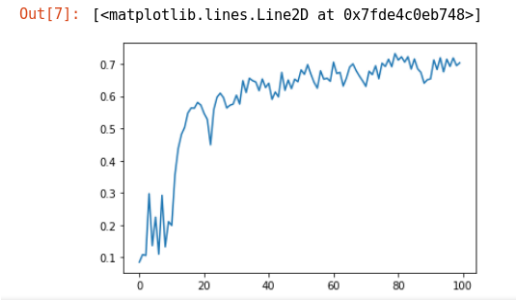
hidden=20, learning_rate=0.001
Accuracy = accuracy: 0.5698333333333333



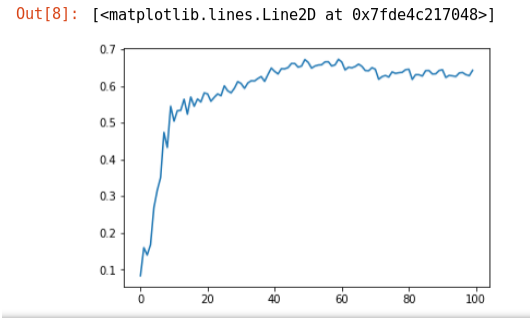
hidden=80, learning_rate=0.001
accuracy: 0.6546666666666666



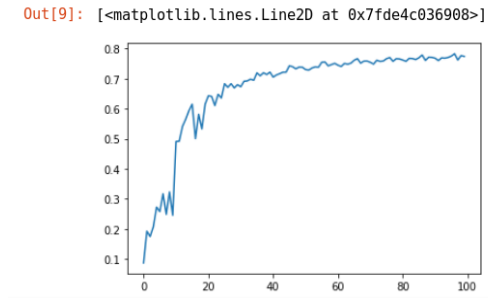
hidden=300, learning_rate=0.001
Accuracy = 0.703



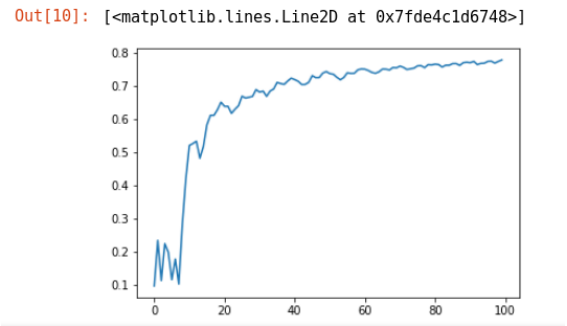
hidden=20, learning_rate=0.0001
Accuracy = 0.6428333333333334



hidden=80, learning_rate=0.0001
Accuracy = 0.7736666666666666

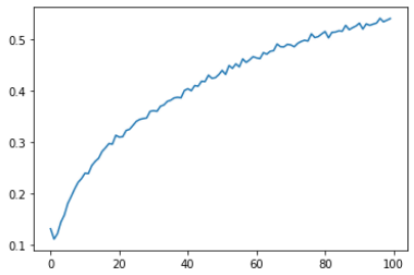


hidden=300, learning_rate=0.0001
Accuracy = 0.7786666666666666



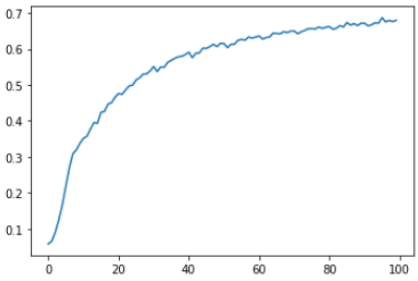
hidden=20, learning_rate=0.00001
Accuracy = 0.5415

```
Out[11]: [<matplotlib.lines.Line2D at 0x7fde4bebe588>]
```



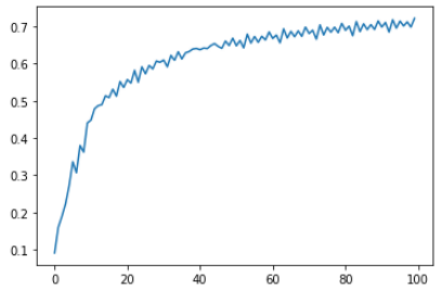
hidden=80, learning_rate=0.00001
Accuracy = 0.6795

```
Out[12]: [<matplotlib.lines.Line2D at 0x7fde4c1603c8>]
```



hidden=300, learning_rate=0.00001
Accuracy = 0.7215

```
Out[13]: [<matplotlib.lines.Line2D at 0x7fde4bdffc88>]
```



4.2 Multi-Layer Neural Network with Keras

4.21 Data

The model is constructed with 784 input nodes, 100 hidden layer nodes (which is a hyper parameter that we can adjust later), and 10 output nodes since there are 10 possible outcome. (784 x 100) and (100 x 10) weights are initialized in the range between -1 and 1, and the biases for each layer is initialized to 0. 'Sigmoid' and 'relu' will be used for the hidden layer's activation function, which is also a hyper parameter. The other hyper parameters include learning rate, learning rate descent, epoch ,and batch size.

4.22 Feed Forwarding

We multiply the parameters with the corresponding attributes and sum them all. The result is then used in the activation function. The output of the hidden layer will become the input of the second layer, and the matrix multiplication training process continues till the training reaches to the output layer. For the output layer, the activation function is softmax, which assigns probability for each of the 10 estimated outcomes the model makes. What is different from the previous model is that it has more layers.

4.23 Back Propagation

To update our parameters, we compare our predictions with the actual values. In order to do that, we need to convert the target to a matching matrix identical to the prediction.

One-hot method is used for the conversion. Throughout the training, the error is monitored since it indicates how well the model is trained and its training direction. To update the second layer's parameters, we multiply the error with the output of the first layer, which will give us a matrix with a dimension identical to the second layer's parameter. Then the parameters are updated by subtracting the multiplication of the matrix and the learning rate, another hyper parameters for this model. The parameters from the first layer are updated with similar method. The error from the output layer is back-propagated to the hidden layer by multiplying the error with the second layer's parameters, and then multiply the result with input layer's data. It will give us a matrix with a dimension identical to the parameters matrix in the first layer, and then we update the parameters again by subtracting the multiplication of the resulting matrix with learning rate. What is different from the previous model is that it has more layers.

4.24 Validation

After epoch times of feed-forwarding and back propagation, to validate the usability and the efficiency of the updated parameters, we use the validation set to predict the values using the updated parameters. And then we calculate the Accuracy.

4.25 Testing All the Hyper-Parameters Combinations.

The following are the hyper parameters that we will test:

hiddens = [[500,100],[300,50],[500,300,100]]
batch_size = [600,5000,10000]

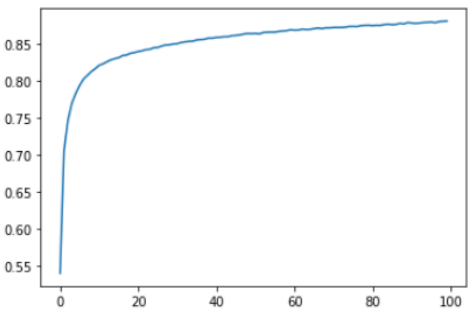
To find the best hyper parameters, we test out each of them.

4.16 Result

The following are all the hyper-parameters combinations and the accuracy scores:

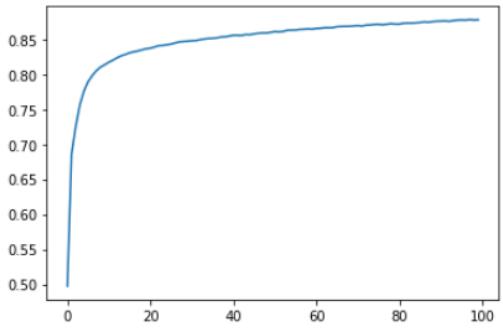
hiddens = [500,100], batch_size = 600
Accuracy = 0.8807, val_acc = 0..8798

```
Out[14]: [<matplotlib.lines.Line2D at 0x7fde2cfbf048>]
```



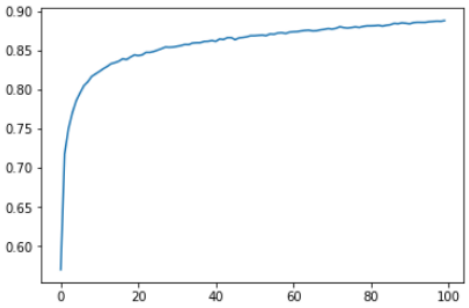
hiddens = [300,50], batch_size = 600
Accuracy = 0.8777, val_acc = 0.8757

```
Out[15]: [<matplotlib.lines.Line2D at 0x7fde28575748>]
```



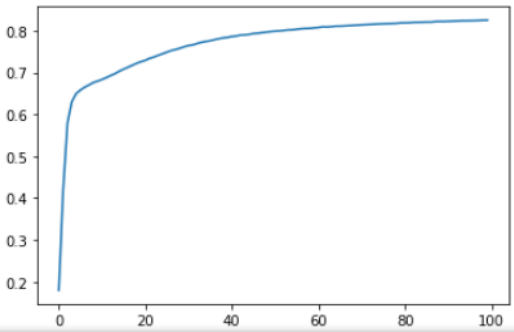
hiddens = [500,300,100], batch_size=600
Accuracy = 0.8878 , val_acc: 0.8862

Out[16]: [<matplotlib.lines.Line2D at 0x7fde2cc10c88>]



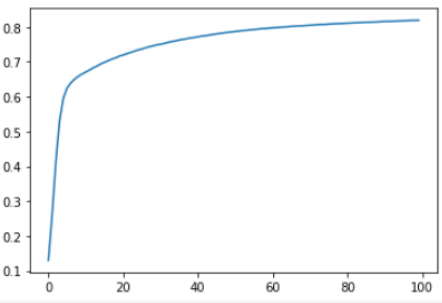
hiddens = [500,100], batch_size = 5000
Accuracy = acc: 0.8255, val_acc: 0.8224

Out[17]: [<matplotlib.lines.Line2D at 0x7fde2c99b438>]



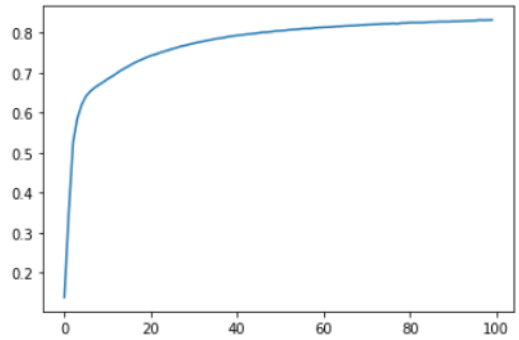
hiddens = [300,50], batch_size = 5000
Accuracy = acc: 0.8190, val_acc: 0.8207

Out[18]: [<matplotlib.lines.Line2D at 0x7fde2c6ff358>]



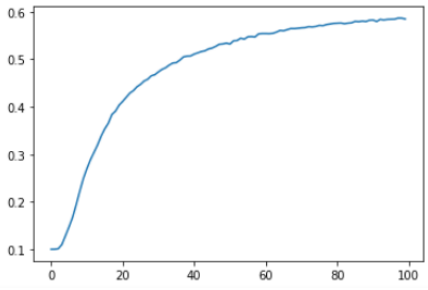
hiddens = [500,300,100], batch_size=5000
Accuracy = acc: 0.8317, val_acc: 0.8360

Out[19]: [<matplotlib.lines.Line2D at 0x7fde2c4556a0>]



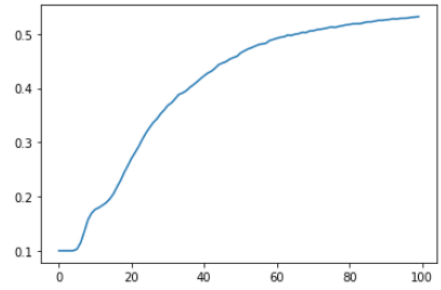
hiddens = [500,100], batch_size = 10000
Accuracy = acc: 0.5849 -val_acc: 0.5882

Out[21]: [<matplotlib.lines.Line2D at 0x7fde2bfcec88>]



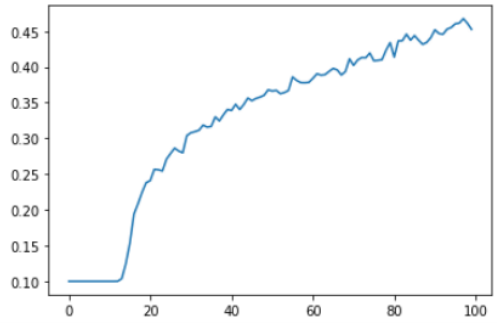
hiddens = [300,50], batch_size = 10000
Accuracy = acc: 0.5318 - val_acc: 0.5353

Out[22]: [<matplotlib.lines.Line2D at 0x7fde2bd4cf98>]



hiddens = [500,300,100], batch_size=10000
Accuracy = acc: 0.4527 val_acc: 0.4626

```
Out[23]: [<matplotlib.lines.Line2D at 0x7fde2ba795c0>]
```



4.3 Convolution Neural Network with Keras

4.31 Data

The model is constructed with input, convolution layer, pooling layer, fully connected layers, and lastly the output layer. The input is structured to take (28,28,1) dimension, the layers in the middle are either convolution layers or pooling layers. The advantage of using these layers is that it allows sharing parameter, which generalizes the pixel with the neighboring pixels, and thus saves time while training. When both the convolution and pooling layers end, the model flattens the output from the previous layer and makes it to a 1 dimension input / output, like neural network model from above. Similar to the models above, this model also make prediction with softmax method on the output layer.

4.32 Architecture

The architecture of the model consist of:
input (convolution layer), convolution layer, max pooling layer, a dense flattened layer, and the output layer.
Filters = 3 & 256, kernel size = 7, activation function = relu, pooling size = 2, dense unit after pooling = 64

4.32 Training

Starting from the input layer, we choose a specific size of kernel and a specific number of filters to begin the convolution. The output will almost always have more layers and less square size. For pooling, we specify the pooling size, and operate either a max pooling or average pooling, which also decreases the size of input / output. When the convolution and pooling steps are done, the output from the step is flattened, making it a 1 dimension input/output. From this point, we repeat the feed forwarding step that we've done for the neural network models above, until it reaches the softmax output layer.

4.34 Validation

After epoch times of training, to validate the usability and the efficiency of the updated parameters, we use the validation set to predict the values using the updated parameters. And then we calculate the Accuracy.

4.35 Result

CNN model was also trained with different data set size and different hyper parameters, but due to the slow training time, only the result from the following hyper parameter was able to obtained:

epoch = 600
batch_size = 100

After an average training time of 80 seconds each epoch, the results are:

```
Epoch 1/600
60000/60000 [==============================] - 85s 1ms/sample - loss: 2.3032 - acc: 0.1012
- val_loss: 2.3026 - val_acc: 0.1016
Epoch 2/600
60000/60000 [==============================] - 79s 1ms/sample - loss: 2.3029 - acc: 0.0980
- val_loss: 2.3025 - val_acc: 0.1027
Epoch 3/600
60000/60000 [==============================] - 78s 1ms/sample - loss: 2.3028 - acc: 0.1006
- val_loss: 2.3027 - val_acc: 0.1012
Epoch 4/600
60000/60000 [==============================] - 78s 1ms/sample - loss: 2.3028 - acc:
0.1008 - val_loss: 2.3026 - val_acc: 0.1035
.
.
.
.
.
Epoch 170/600
60000/60000 [==============================] - 75s 1ms/sample - loss: 0.4007 - acc: 0.8348
- val_loss: 0.0820 - val_acc: 0.9747
Epoch 171/600
60000/60000 [==============================] - 75s 1ms/sample - loss: 0.3942 - acc: 0.8387
- val_loss: 0.0820 - val_acc: 0.9736
Epoch 172/600
60000/60000 [==============================] - 75s 1ms/sample - loss: 0.3966 - acc: 0.8375
- val_loss: 0.0756 - val_acc: 0.9753
Epoch 173/600
60000/60000 [==============================] - 75s 1ms/sample - loss: 0.3926 - acc: 0.8391
- val_loss: 0.0777 - val_acc: 0.9762
Epoch 174/600
60000/60000 [==============================] - 78s 1ms/sample - loss: 0.3868 - acc: 0.8418
- val_loss: 0.0817 - val_acc: 0.9758
Epoch 175/600
60000/60000 [==============================] - 78s 1ms/sample - loss: 0.3886 - acc:
0.8415 - val_loss: 0.0727 - val_acc: 0.9762
```

As we can see, the accuracy started from very low (10%) and gradually increased to 84%. For the validation data, it is raised up to 97.6%

6. Conclusion

As we can see from the results, the first model and the second model gives the best result when the learning rate and the number of hidden nodes are in the moderate / average level. For the convolution model, the results were very similar most of the time, except that the training time varied based on the data set size, batch size, kernel size, filters, etc.
When all three models were tested with the testing data, the results were in the range of -2 and +2 of the accuracy from the training, which means that all models were properly trained without any overfitting or underfitting.

The categorizing using the optimal parameters and hyper-parameters mostly gave accuracy score between the range of 0.7 and 0.9. Although the concept of Neural Network was not so easy to understand, in the future work I would like to test this model on more data set.