Introduction: Summary and problem definition for management

Assignment 6 builds on the analysis of the last assignment, and requires the application of an artificial neural network (ANN) for prediction of handwritten digits in the MNIST dataset. In this assignment, we apply the TensorFlow library instead of SciKit Learn's Random Forest Classifer. As before, the challenge is to build a model that successfully assigns a digit that is equal to the handwritten one. The challenge is to build a model that accurately distinguishes between each digit, while taking into account the time it takes to train and test the data.

Assignment 6 – Artificial Neural Networks (MNIST)

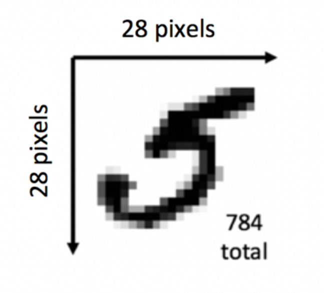
***MNIST** – Modified National Institute of Standards and Technology (example data)*

The steps for the analysis are the following:

1) Build a 'Deep Neural Network' (DNN) model using TensorFlow
2) Train & test various designs of the model using 'hidden layers'
3) Time the model build and record the train/test accuracy
4) Compare the model timing, Precision & Recall Scores

Research Design, Measurement & Statistical Methods

The data set for this assignment is the same as the previous one. The MNIST data set consists of 70,000 handwritten digits, and is a popular example of training machine learning models. Each row represents one hand written example using 785 variables. The first column is the 'response' variable, which is the actual value to test the predicted estimate against. The remaining 784 columns are the integer grey scale values of each pixel in a 28 x 28 pixel square .



An example of a plotted row of data (784 pixels)

For example, the plot on the left is a binary plot showing a row of data that has a y value of '5'. The first column of data is the actual value – for training & testing. The subsequent 784 columns of data are the greyscale values for each of the 28x28 pixels representing the digit.

The next step in the process is to split the data by training & test data sets. In the instructions, we are told to use 60,000 rows for training, and 10,000 for test. I then shuffle the training data set by the index to randomize the data.

We then begin the experiment by creating multiple deep neural networks using TensorFlow.

Model overview:

Below is a summary of the models that were created in the process of this analysis and their performance comparison. Performance variables include the reduced number of estimators as well as the accuracy scores of training & test sets. In the next section, I will review the build for each model and a visual comparison of each models' 'confusion' matrix.

	Model 2a (2 layers x 10 neurons)	**Model 2b (2 layers x mixed neurons)**	**Model 5a (5 layers x 20 neurons)**	**Model 5b (5 layers x mixed neurons)**
Model Parameters & Performance	DNNClassifier hidden_units=[10,10] Accuracy (train/test) .94/.93	DNNClassifier hidden_units=[150,50] Accuracy (train/test) .94/.97	DNNClassifier hidden_units=[20,20,20,20,20] Accuracy (train/test) .96/.94	DNNClassifier hidden_units=[300,200,100,50,25] Accuracy (train/test) 1.00/.975
	Time to build: **~100 sec**	Time to build: **~184 sec**	Time to build: **~184 sec**	Time to build: **~305 sec**

Programming overview:

Below is the code that was used to develop the various DNNClassifier models, and a snapshot of the confusion matrix:

Model 2a

```
feature_cols = [tf.feature_column.numeric_column("X", shape=[28 * 28])]
dnn_clf_2a = tf.estimator.DNNClassifier(hidden_units=[10,10],
                    n_classes=10, feature_columns=feature_cols)
input_fn_2a = tf.estimator.inputs.numpy_input_fn(x={"X": X_train},
      y=y_train, num_epochs=50, batch_size=50, shuffle=True)
dnn_clf_2a.train(input_fn=input_fn_2a)
```

```
Confusion Matrix model 2a:
[[ 946    0    2    1    2    7   13    6    1    2]
 [   0 1122    2    2    1    1    2    2    3    0]
 [  17   10  932   21    9    2    9    9   22    1]
 [   2    0   26  916    2   25    1   14   17    7]
 [   2    2    6    1  930    1    6    4    3   27]
 [   6    4    3   21    8  804   10    8   22    6]
 [  10    3    8    0    9   17  907    0    2    2]
 [   3    7   18    5    3    0    0  958    4   30]
 [   6    3    7   22    3   24    8    8  884    9]
 [  10    6    0   11   17    7    0   17   12  929]]
```

Model 2b

```
feature_cols = [tf.feature_column.numeric_column("X", shape=[28 * 28])]
dnn_clf_2b = tf.estimator.DNNClassifier(hidden_units=[150,50],
                    n_classes=10, feature_columns=feature_cols)
input_fn_2b = tf.estimator.inputs.numpy_input_fn(x={"X": X_train},
      y=y_train, num_epochs=50, batch_size=50, shuffle=True)
dnn_clf_2b.train(input_fn=input_fn_2b)
```

```
Confusion Matrix model 2b:
[[ 968    0    2    0    1    1    3    2    2    1]
 [   0 1122    3    0    0    3    3    2    2    0]
 [   2    2 1007    4    4    0    2    9    2    0]
 [   0    0    3  988    0    5    0    3    5    6]
 [   1    0    1    0  961    0    6    3    2    8]
 [   0    0    0    9    1  872    5    1    2    2]
 [   3    3    1    0    5    5  937    0    4    0]
 [   1    1    7    1    1    0    0 1008    4    5]
 [   5    0    1    5    3    5    1    3  948    3]
 [   2    3    0    5    7    3    0    4    5  980]]
```

Model 5a

```
feature_cols = [tf.feature_column.numeric_column("X", shape=[28 * 28])]
dnn_clf_5a = tf.estimator.DNNClassifier(hidden_units=
     [20,20,20,20,20], n_classes=10, feature_columns=feature_cols)
input_fn_5a = tf.estimator.inputs.numpy_input_fn(x={"X": X_train},
      y=y_train, num_epochs=50, batch_size=50, shuffle=True)
dnn_clf_5a.train(input_fn=input_fn_5a)
```

```
Confusion Matrix model 5a:
[[ 959    0    2    1    2    8    1    4    2    1]
 [   0 1112    5    3    0    1    2    2    9    1]
 [   5    4  959   16    5    9    8   11   14    1]
 [   1    0   11  947    0   26    0   11   11    3]
 [   2    2    5    0  942    0   11    2    1   17]
 [   5    4    1   16    2  827    6    5   20    6]
 [   9    2    2    0    8   12  921    0    4    0]
 [   1    6   12    7    7    4    0  969    2   20]
 [   6    3    4   16    6   19    2    3  906    9]
 [   8    0    0    7   31    6    0   11   11  935]]
```

Model 5b

```
feature_cols = [tf.feature_column.numeric_column("X", shape=[28 * 28])]
dnn_clf_5b = tf.estimator.DNNClassifier(hidden_units=
   [300,200,100,50,25], n_classes=10, feature_columns=feature_cols)
input_fn_5b = tf.estimator.inputs.numpy_input_fn(x={"X": X_train},
      y=y_train, num_epochs=50, batch_size=50, shuffle=True)
dnn_clf_5b.train(input_fn=input_fn_5b)
```

```
Confusion Matrix Model 5b:
[[ 967    1    1    0    2    1    3    1    3    1]
 [   0 1123    5    0    0    0    2    1    4    0]
 [   5    1 1005    5    3    0    2    8    3    0]
 [   0    0    3  985    0    5    0    2    5   10]
 [   1    1    2    0  958    0    5    5    1    9]
 [   2    0    2   11    2  859    3    0    8    5]
 [   3    2    0    0    4    7  938    0    4    0]
 [   1    3    7    3    0    0    0 1006    3    5]
 [   5    0    2    9    3    3    3    3  943    3]
 [   2    2    0    8   10    4    0    5    5  973]]
```

Conclusion: The recommendation is to use model 5b. Although it takes longer to train than the other models (~5 mins), the accuracy of the test predictions is much higher than the other models. In the end, the 5 minutes that it takes to train the model is much faster than the models developed for the previous assignment. For example, to achieve similar accuracy levels using RandomForestClassifier in Assignment #5, it took close to 27 minutes. In that exercise, we settled for an accuracy of ~.9 for a model that took close to 10 minutes to build and test. Using the DNNClassifier method in TensorFlow, we were able to achieve much higher levels of accuracy in a fraction of the time.

Note: A shared version of the python notebook used to create this analysis can be found at Google CoLab:

https://colab.research.google.com/drive/1C0vGy2ANuZu8ZSevwag1ydboKfS1JmMc