Summary and Problem Statement

In this exercise, is a benchmarking experiment to compare aspects of artificial neural networks doing classification of the MNIST data set. Multiple neural networks were created to fit models to the MNIST data. The differences between models, created by adding more nodes and adding more layers, are recorded. Total time to build the model and evaluate the results are collected. Model "goodness" is assessed by comparing accuracy over the set of test data.

Methodology

I was not able to figure out how to modify the Google TensorFlow code samples to increase the number of nodes per layer, so I worked off material from the course text[1], specifically, the example on pages 265 – 272[1] which creates a neural network using the TensorFlow API. I also created a Scikit-Learn version of the program on page 264[1] for comparison. For simplicity, I ran each program multiple times, changing the number of nodes and layers by commenting and uncommenting various values. After each run, the required output was saved. To ensure a clean comparison, during runs all applications and windows were closed except the Python IDE, and the Python kernel was restarted before each run.

Code Overview

The code loads the MNIST data set, defines, then runs the neural network models. Variables were created for the number of nodes per layer, and a placeholder for number of layers was created to facilitate tracking. In both programs, the sample code from the text forms the base of the work. Timers were added to capture time to train the network and time to make predictions, these are summed to get total time. For visualization, and the end of each run the confusion matrix is generated to see the model's per-digit performance on the test data set.

Results and Observations

The results from the runs are summarized in the table below[2]:

Table 1 – Results from multi-layer neural network benchmark testing

API	Number of layers	Nodes per Layer	Total Time (sec)	Accuracy - Train data	Accuracy - Test data
TF direct	2	10	28.21199	0.98	0.9368
TF direct	2	50	37.09026	1	0.9686
TF direct	5	10	34.32393	1	0.9346
TF direct	5	50	53.11795	1	0.9708
Scikit-compatible	2	10	38.78268	0.9525	0.9385
Scikit-compatible	2	50	47.96781	0.9998	0.9763
Scikit-compatible	5	10	46.41774	0.9522	0.9355
Scikit-compatible	5	50	59.2583	0.9999	0.9739

As expected, the larger the number of layers the longer the run, ditto the number of nodes. The time for adding layers versus nodes appear to be roughly equal when amortized across the total number of nodes in all layers[2]. In both cases adding layers without adding nodes caused test accuracy to drop. In all cases adding nodes to existing layers improved accuracy on the test data. For the TensorFlow model, adding both layers and nodes gave the best results on the test data. For the Scikit-Learn model the 2-layer 50-node model did better than the 5-layer 50-node one. I cannot account for this unless it is a side-effect of over-training, which is possible give the training accuracy. In all cases, the difference between training and test accuracy is ~.02. The 50 node models all appear to have possible over-training issues given the high training accuracy. In conclusion, adding nodes seems to improve accuracy, adding both nodes and layers seems to be the best way to improve a neural network.

[1] *Hands-on Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Geron, ch. 10
[2] Metric comparison.xlsx