Summary

First proposed in the 1944 by researchers to model and understand the biological counterpart better, artificial neural networks (ANNs) helped to form the basis of deep learning. In today's society, ANNs are playing a growing role in computer vision, where ANNs are able to take in large amounts of input and process them to suggest complex relationships that might have as well been hidden to other algorithms. Within computer vision, the abilities of ANNs can be applied to image and character recognition (such as handwritten characters).

Research Design

For this assignment, we looked at the popular dataset from the MNIST database that is widely used in the field of machine learning. The MNIST dataset containing 70,000 examples of handwritten digits of 0-9. With the use of classifiers, we can separate the data into a training (60,000 examples) and test set (the remaining 10,000 examples) to see how well, or the probability the classifier model can successfully distinguish a handwritten digit into the correct associated number to it using 784 explanatory variables.

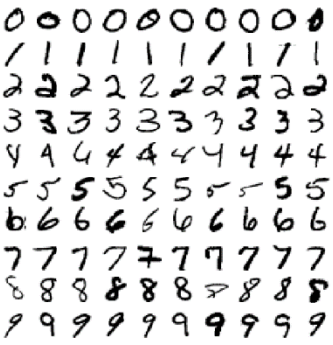

Figure 1: Sample images from the MNIST dataset

Technical Overview

In order to conduct the classification performance experiment, we will carry out a factorial design of two levels on two experimental factors. We also will be making use of a class of feed-forward ANNs called the Multilayer Perceptron (MLP), which consists of three layers of nodes, the input, hidden, and output layers. With the structure of the MLP in mind, the two experimental factors will include two different values for the number of nodes per hidden layer (20, 50) and

the number of hidden layers used (2, 5). For classification, we will compute the softmax cross entropy between logits and the model will be optimized using the Adam Optimization Algorithm (similar to the classical stochastic gradient descent) because it is computationally efficient. The experiment will be conducted using a Python API for TensorFlow, which is an open source machine learning library, initially developed by Google. The metrics we will be comparing will be the model performance as well as model runtime (in seconds).

Results

	# of Hidden Layers	Nodes per Layer	Processing Time (seconds)	Test Set Accuracy
Model A	2	20	0.588576	0.8582
Model B	2	50	0.564457	0.8562
Model C	5	20	0.594847	0.8151
Model D	5	50	0.690888	0.8586

For our benchmark model, we started off with 2 hidden layers and 20 nodes. For the following models we experimented with changing the hidden layers amount as well as the number of nodes per layer. In terms of processing time, time increased as the number of layers/nodes increased, with Model D taking the longest at 0.69 seconds – which is very reasonable still. For test set accuracy, the models performed very closely to one another, with Model D performing the best at 85.86% accuracy. I would recommend using the benchmark model Model A as processing time is ~0.10 seconds less than Model D, but with only a trade off of 0.06. In general, it is a good idea to keep the number of nodes as low as possible for the model to generalize well. In addition, it can be said that it is hard to find reason to increase the number of hidden layers beyond two layers. Therefore, Model A is a good candidate.