**Building a neural network to predict the digit in an image**

*Sekhar Mekala*

Please visit <https://github.com/cuny-sps-msda-data622-2017fall/project-msekhar12/blob/master/MNIST%20Project.ipynb> for a detailed description of the model building process.

The following options were used to train and build the neural network:

1. **Scaling:** The elements (pixels) were scaled using the standard scaler method. The standard scaling transformation will help to make all the inputs to a similar scale, and it will help the gradient descent algorithm to converge to an optimal value quickly.
2. **Activation function:** The *relu* activation function has been chosen, since the validation accuracy obtained using *relu* is higher than the *tanh* and *sigmoid* accuracy. I divided the 70000 observations into 60000 training observations and 10000 test observations. Out of the 60000 training observations 6000 were randomly picked (stratified on the digit) as validation data. A preliminary model is built with 40 nodes as the first layer, and 10 nodes as the final layer using the 54000 observations. This has given 94.6%, 94.5%, and 96.45% validation accuracies for *sigmoid, tanh* and *relu* respectively*.* So *relu* activation was chosen to train the model.

In general, *relu* is a better activation function as it helps us to get higher derivative values as compared to *sigmoid* and *tanh.* The *sigmoid* and *tanh* activations have a very small derivative value at the extremes, which will make the gradient descent to converge very slow (although *tanh* is better when compared to *sigmoid*).

1. **Other options:**

Chosen 4 layers (including the input layer). The first layer (input layer) consists of 784 input values (standardized pixel values). The second layer consists of 128 nodes, third layer of 40 nodes and final layer of 10 nodes. A dropout value of 0.2 was chosen. 150 epochs were chosen. A batch size of 1000 was used (this is needed to perform Stochastic Gradient Descent).

Stochastic Gradient descent will help to speed up the training by rigorously updating the weights with randomly chosen data (batch size of 1000 observations). This results in faster convergence.

The remaining parameters (number of layers etc.) were chosen, as the validation score was the best. A summary of all the options tested, along with the validation accuracy scores are given below:

* Set-1
  + epochs=50
  + batch\_size = 1000
  + First layer nodes = 128
  + Last layer nodes = 10
  + Activation = 'relu'
  + *Validation score: 0.9711666666666666*
* Set-2
  + epochs=50
  + batch\_size = 1000
  + First layer nodes = 40
  + Second layer nodes = 15
  + Last layer nodes = 10
  + Activation = 'relu'
  + *Validation score: 0.9586666666666667*
* Set-3
  + epochs=150 (More epochs due to dropout)
  + batch\_size = 1000
  + First layer nodes = 128
  + Last layer nodes = 10
  + Dropout = 0.2
  + Activation = 'relu'
  + *Validation score: 0.9705*
* Set-4
  + epochs=150 (More epochs due to dropout)
  + batch\_size = 1000
  + First layer nodes = 128
  + Second layer nodes = 40
  + Last layer nodes = 10
  + Activation = 'relu'
  + *Validation score: 0.9736666666666667*

Set-4 options were chosen and this has given a test accuracy (NOT validation accuracy) of 97.74% (as shown in the ipython notebook.).

Convolutional neural networks were not used for this project, since the given digits are of the same size, almost concentrated at the center of the image. So our neural network might fail to detect images of different sizes scattered over the image in different sizes.