CSE 40625 — Machine Learning
Assignment 5 (10 points)
Due Date: April 20, 2017 (Sakai Drop Box)

Multi-Layer Neural Network

Overview
The purpose of this assignment is for you to implement a multi-layer neural network, a powerful supervised machine learning inspired by the way the brain works. Multi-layer neural networks are systems of interconnected "neurons" or units that can compute values from inputs by feeding information through the network. The model makes predictions by chaining together layers of neural units, with the output of neurons at earlier layers used as the input to neurons at later ones. Materials for the assignment, including the dataset, expected output, and template code can be found here on GitHub.

You may use Python libraries for handling data preprocessing and visualization, including but not limited to NumPy, SciPy, pandas, and Matplotlib, but *you may NOT use any Python libraries that employ machine learning models, including but not limited to scikit-learn, StatsModels, TensorFlow, or Orange*. Your solution to the assignment should be individually submitted.

Dataset
You will use the "digits" dataset on handwritten digit classification with all 10 classes (labeled from 0 to 9) for this assignment. The data is provided in comma-separated (CSV) file format. For all rows, the last column designates the class (y) and the remaining columns designate features (X). The first row consists of the feature and class names.

Procedure
Use the above digits dataset as input to a multi-layer neural network model with two hidden layers of 100 units each. Initialize the weights and bias at each layer with random samples using normalized initialization. Initialize the activations as input x. Train the network with batches of 100 instances at a time. For each batch of inputs x, feed the activations of the batch forward through the network. At each hidden layer, apply the hyperbolic tangent function to the activations plus bias to compute the activations for the next layer. Use the softmax function to compute the activations for the output layer. Using cross-entropy loss, compute the sensitivities of the final layer as twice the difference of the final activations and the target values. Propagate the sensitivities backwards, using the sensitivities at the current layer to compute the sensitivities at the previous layer. Update the weights at each layer using the minimum gradient of the error function with a learning rate of 0.01. To reduce overfitting, apply an L2 regularization term to the gradients parameterized with a lambda of 1.0 before using the gradients to update the weights. Repeat for 500 iterations. Predict each target value based on the unit in the final (output) layer with the highest activation output.

Output
Your code should output the accuracy of the prediction results every 50 iterations while fitting the model, a blank line, and the confusion matrix of the final output results.

Example output:

```
0 0.745
50 0.986
100 0.987
150 0.987
200 0.988
250 0.988
300 0.988
350 0.988
400 0.988
450 0.988

Predicted   0   1   2   3   4   5   6   7   8   9
Actual
0         552   0   0   0   1   0   1   0   0   0
1           0 567   0   0   0   0   0   1   1   2
2           1   1 553   1   0   0   1   0   0   0
3           0   0   1 566   0   3   0   0   0   2
4           1   0   0   0 561   0   2   0   2   2
5           0   0   2   2   0 549   0   0   0   5
6           0   3   0   0   2   0 553   0   0   0
7           0   0   0   1   0   0   0 564   0   1
8           0  10   0   1   1   0   0   0 542   0
9           1   4   0   2   2   2   0   1   2 548
```

Submission
Please submit a Python executable (multilayer_neuralnet.py) file of your code to your Sakai Drop Box.

Should you run into any problems, please feel free to email or meet with the instructor.