Assignment: Project 1
Course: CS557 Computational Intelligence and Machine Learning
Instructor: Dr. Razvan Andonie
Student: Andrew Dunn

Introduction

For this project we created a program written in Python that can classify an input matrix of 1's and 0's to determine if the 1's form the shape of the letter 'I' or the letter 'L'. To do this, we implemented a basic single neuron perceptron model that can "learn" the correct input patterns through a process called training. This training process entails processing some finite input set of unique 'I' and 'L' patterns that are represented as a 2-dimentional matrix of 1's and 0's. The model contains a vector of weight values that, when multiplied by the input matrix, determines the model prediction after the result is passed through a sigmoid function. This weight vector is adjusted dynamically during the training process, with the goal of finding a linear separation line between the two 'I' and 'L' input classes. If the model successfully finds this separation line, then this is denoted as the model "converging" and the training is ended. It is possible however that the two input classes are not linearly separable. When this happens, the model will not converge, and thus the training process will continue indefinitely. Since we humans do not have an infinite amount of time to wait, and upper limit of 1,000 iterations has been set for the training process. After 1,000 iterations the training process will halt regardless if convergence has been reached. A single iteration is defined as a single pass through the entire input set. In addition to the maximum iterations, a fixed learning rate of 0.1 has been used in all experiments. The learning rate determines the magnitude in which the weight vector values are adjusted at each training step. Typically, a smaller learning rate is preferred to prevent overshooting of the linear separation line.

Input Data

The input data containing the 'I' and 'L' samples is randomly generated at run-time within certain constraints. We tested four different sample variations, each containing slightly different shapes for the 'I' and 'L' characters. A description of these sample variations can be found in Table 1 below. The values contained within Table 1 represent a length in pixels. A single pixel in the input matrix is represented as a cell containing a 1. Empty pixels are represented as cells containing 0's. Here are some example 3x3 matrices containing an 'I' character:

1	0	0
1	0	0
1	0	0

0	1	0
0	1	0
0	1	0

0	0	1
0	0	1
0	0	1

Figure 1: Example 'I' Input Samples

And here are some example matrices containing an 'L' character:

1	0	0
1	0	0
1	1	0

0	1	0
0	1	0
0	1	1

Figure 2: Example 'L' input samples

Note that the given example matrices in Figures 1 and 2 contain the entire input data set for the 3x3 matrix with sample variation 1, which is the first data row in Table 2 below. With larger matrix sizes, the shapes and sizes of the ‘I’ and ‘L’ characters remain the same, but the location within the matrix changes between different samples in the same set.

Table 1: Sample Variation Set Descriptions						
Sample Variation	Min. I Height	Max I Height	Min. L Height	Max L Height	Min. L Width	Max L Width
1	3	3	3	3	2	2
2	2	3	2	3	2	3
3	3	4	3	4	2	3
4	3	5	3	5	2	4

Results

In this section you will find the results of our experiments. We tested the four different sample variations in Table 1 with several different matrix sizes. Table 2 shows which of these experiments successfully converged and thus was able to find a linear line of separation between the two ‘I’ and ‘L’ classes. If an experiment failed to converge, the accuracy of the training sample set is displayed as a percentage less than 100%. This percentage represents the portion of training input samples that were successfully classified with the resulting model after 1,000 training iterations. The remaining portion of samples were incorrectly classified, and thus a linear separation line for these samples were not found. Note that the *Sample Size* column in Table 2 is the sum count of all ‘I’ and ‘L’ samples for that variation.

As shown in Table 2, all 4 sample variations were tested on matrix sizes ranging from 3x3 to 10x10. The 3x3 matrix and 4x4 matrix are missing some sample variations because they could not fit in the limited size of the matrix. In addition to these experiments, we tested a few larger matrix sizes ranging up to 100x100 using sample variation 1 to see if it continues to converge, which they do. In fact, sample variation 1 is the only variation that converges for every matrix size tested. This is most likely due to the strict size restrictions for the ‘I’ and ‘L’ characters. In this sample variation, all ‘I’ and ‘L’ characters are exactly 3 pixels tall, and the ‘L’ characters are all 2 pixels wide at the bottom stroke. Sample variations 2, 3, and 4 allow more differences between the individual sample matrices. These differences make it harder for the model to successfully find a linear separation line between the two classes.

Table 2: Convergence Success and Training Set Accuracy				
Matrix Size	Sample Variation	Sample Size	Converges	Train Accuracy
3x3	1	5	Yes	100%
3x3	2	17	Yes	100%
4x4	1	14	Yes	100%
4x4	2	45	No	84.4%
4x4	3	27	Yes	100%
5x5	1	27	Yes	100%
5x5	2	81	No	81.4%
5x5	3	59	Yes	100%
5x5	4	83	Yes	100%
6x6	1	44	Yes	100%
6x6	2	134	No	84.3%
6x6	3	104	No	88.5%

6x6	4	158	No	81.7%
7x7	1	65	Yes	100%
7x7	2	193	No	82.9%
7x7	3	160	No	85.6%
7x7	4	257	No	79.0%
8x8	1	88	Yes	100%
8x8	2	267	No	81.2%
8x8	3	228	No	89.9%
8x8	4	367	No	81.2%
9x9	1	118	Yes	100%
9x9	2	346	No	86.7%
9x9	3	300	No	84.7%
9x9	4	517	No	79.9%
10x10	1	148	Yes	100%
10x10	2	436	No	83.5%
10x10	3	382	No	85.1%
10x10	4	661	No	80.8%
15x15	1	374	Yes	100%
20x20	1	686	Yes	100%
25x25	1	1059	Yes	100%
30x30	1	1538	Yes	100%
40x40	1	2713	Yes	100%
50x50	1	4329	Yes	100%
100x100	1	10673	Yes	100%

Next, we will look at the model's ability to generalize on unseen data, and the effect that random noise has on the testing accuracy. You can find the results of these experiments in Table 3 below. The sample input sets were randomly generated and then split into 80% training and 20% testing data sets. The training and sample data sets are disjoint, and thus do not share any input samples. Note that none of the experiments displayed in Table 3 managed to converge, so the testing accuracy is from the final model after 1,000 iterations. The *# Noise Pixels* column in Table 3 contains the upper limit for how many noise pixels are randomly added to input samples. A noise pixel is a randomly selected matrix cell in an input sample that has been set to a value of 1. In Figure 3 below you can see an 'I' sample with three different configurations of random noise. Note that the red and underlined 1's are the added random noise pixels.

$$\begin{array}{|c|c|c|}\hline \mathbf{1} & 0 & \underline{1} \\ \hline \mathbf{1} & 0 & 0 \\ \hline \mathbf{1} & 0 & 0 \\ \hline \end{array} \quad \begin{array}{|c|c|c|}\hline \mathbf{1} & 0 & 0 \\ \hline \mathbf{1} & \underline{1} & 0 \\ \hline \mathbf{1} & 0 & \underline{1} \\ \hline \end{array} \quad \begin{array}{|c|c|c|}\hline \mathbf{1} & \underline{1} & 0 \\ \hline \mathbf{1} & 0 & \underline{1} \\ \hline \mathbf{1} & \underline{1} & 0 \\ \hline \end{array}$$

Figure 3: Example 'I' Input Samples with Added Noise Pixels

As you can see in Figure 3, the base input sample is the same in all three examples: An 'I' character in the left most column. The random noise pixels create additional variations of the same input sample, thus increasing the size of the training and testing sample sets. This effect can be seen in Table 3 below. For a single matrix size, as the number of noise pixels increase so does the size of the training and testing input sets.

Table 3: Effect of Noise Pixels on Testing Accuracy (Sample Variation 2)

Matrix Size	Training Size	Testing Size	# Noise Pixels	Testing Accuracy
3x3	14	4	0	25.0%
3x3	79	20	1	85.0%
3x3	176	45	2	77.8%
3x3	250	63	3	76.2%
4x4	35	9	0	66.7%
4x4	420	106	1	84.9%
4x4	1860	466	2	74.7%
4x4	4818	1205	3	77.7%
5x5	67	17	0	70.6%
5x5	1273	319	1	83.1%
5x5	9250	2313	2	83.1%
5x5	14317	3580	3	79.4%
6x6	108	27	0	70.4%
6x6	3026	757	1	81.1%
6x6	14227	3557	2	78.9%
7x7	153	39	0	66.7%
7x7	6068	1518	1	82.3%
8x8	205	52	0	55.8%
8x8	10518	2630	1	82.5%
9x9	282	71	0	56.2%
9x9	13414	3354	1	84.5%
10x10	352	89	0	68.5%
10x10	14056	3514	1	84.5%

The goal of these experiments was to see if by adding some random noise we could increase the overall testing accuracy. Looking at Table 3, you can see that this is indeed the case. Note that for these experiments, we only used sample variation 2. For all matrix sizes tested, the random noise pixels increased the testing accuracy. Adding more than one noise pixel to the samples did not seem to increase accuracy further in all cases, and in fact decreased the testing accuracy for most matrix sizes. Therefore, adding a single random noise pixel to samples gives the best results in our experiments, and increases the ability for our model to generalize on unseen input samples. This is probably in large part due to the larger training and testing data sets that the random noise gives us.

Conclusions

In this paper we detailed our project and the experiments that we conducted. Despite being a simplistic machine learning model, the single neuron perceptron can still perform well depending on if the input samples can be linearly separable. The models' ability to generalize is weak, but this can be much improved by adding a single random noise pixel to the input samples. This brought our testing accuracy up to around 80% – 85% accuracy for all matrix sizes from 3x3 to 10x10. This was an interesting project and is a great introduction to how neural networks work.