Handwritten Digit Recognition Using Perceptron Neural Network

Yun Lan and Sean Lee

University of California, Irvine
Department of Information and Computer Science
Irvine, California 92697
ylan@ics.uci.edu and seanl@ics.uci.edu

Many interesting problems fall into the general area of pattern recognition and classification. In this report, we have explored a specific area in which artificial neural network was heavily used for recognition of patterns, and would like to present what we have learned and accomplished for this interesting and exciting topic.

Introduction

What is an artificial neural network and how does it work? Artificial neural network have been developed from generalizations of neural biology model, based on the assumptions that (a) information processing occurs at many simple elements called neurons, (b) signals are passed between neurons over connection links, (c) each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted, and (d) each neuron applies an activation function to its net input to determine its output signal. A neural network is characterized by its pattern of connections between the neurons, its method of determining the weights on the connections, and its activation function. A neural net consists a large number of neurons, connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve a problem. Each neuron has an internal state, called its activation or activity level, which is a function of the inputs it has received. When the resulting function from the given inputs exceed a predefined threshold, the neuron "fires" (send signal to other neurons). The neural net just been described can be modeled in the following way: Given a neuron **Y** that receives inputs from neurons $\mathbf{X_1}$, $\mathbf{X_2}$, and $\mathbf{X_3}$. The weights on the connections from $\mathbf{X_1}$, $\mathbf{X_2}$, and $\mathbf{X_3}$ to neuron **Y** are w_1, w_2, and w_3. The input to neuron **Y** is the sum of the weighted signals from neurons $\mathbf{X_1}$, $\mathbf{X_2}$, and $\mathbf{X_3}$ given in the following equation: $y\_in = w_1x_1 + w_2x_2 + w_3x_3$
The activation function of y is then computed based from the input value for neuron **Y**. Figure 1 illustrates the model of our simple neural net (typically called single-layer neural network).

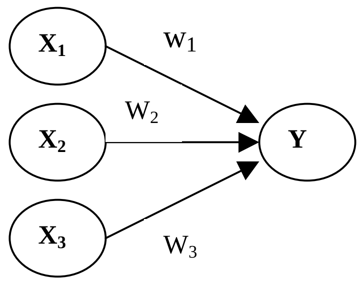

Figure 1 Simple neural net model

The neural network we just described is very simple. It is sufficient for this network to solve most of simple problems. For more complicated problems, hidden units can be added to the network to increase its problem solving ability (Papert and Minsky, 1988). This category of networks, typically called multi-layer nets, takes input from adjacent levels of units and computes the value for its hidden units. The model is illustrated in Figure 2. The computation process is similar to single-layer net, however, in some cases, training may be more difficult (requires backpropagation of weights).

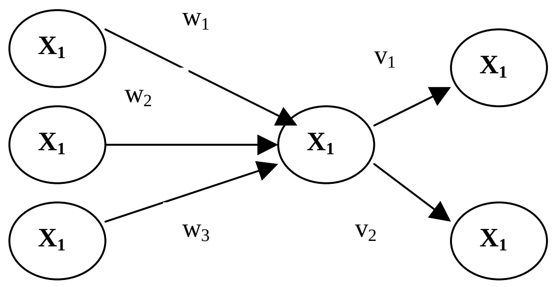

Figure 2 Multi-layer neural net model

The next step after defined neural net model is training. There are two categories for training: supervised and unsupervised. In this paper, we will just focus on the supervised training since it is the method we have chosen in the project implementation. In its simplest form, supervised training is accomplished by presenting training patterns; each associated with a true pattern to the neural net. The weights are then adjusted according to a learning algorithm. The training of neural net and weight adjustment is strongly associated with how we define activation function for the given neuron. As mentioned before, the neuron involves summing its weighted input and applying an output based on an activation function. Typically, all neurons used the same activation function. There are several activation functions are currently used by neural networks: (a) identity function, (b) binary step function, (c) binary sigmoid, and (d) bipolar sigmoid. To keep our discussion within the allotted time frame, we won't go into the description of each activation function. Instead, we will assumed our activation function for the project using a version of binary step function having the following property:

$$f(x) = \begin{cases} 1 \text{ if } x > \text{threshold} \\ 0 \text{ if } x < \text{threshold} \end{cases}$$

For the neural nets we have studies, our intention is to train the net so when present it a pattern that is or similar to any of pattern in the train set, the neural net will be able to give us a "yes" response. A pattern that is not in any resemblance to patterns in the training set should give us a "no" response. Since we want one of two responses, the activation function is taken to be a step function. This is the reason why we pick binary step function to be the activation function for our perceptron. The input of neural net is the summation of weighted input based from the equation: $y\_in = b + \Sigma\, x_i w_i$ where b is the bias to perceptron
The activation function will "fire" (or trigger) when y_in > 0, and remains in the same internal state when y_in < 0. This basically divides the data into two regions, determined by the value of y_in. The above equation can then be rewrite as: $b + \Sigma\, x_i w_i = 0$
Depending on the number of input units in the network, this equation represents a line, a plane, or a hyperplane. The division line separates the "yes" region from "no" region. In a paper written by Minsky and Papert, they showed that a single-layer net could learn only linearly separable problems. It also shown that multi-layer net with linear activation functions are no more powerful than single-layer nets (Minsky and Papert, 1988). In another words, if input patterns were happened to be on the division line or the algorithm is unable to make distinction of two clusters, then the neural net is unable to learn or perform any classification.

We have seen the architecture of the neural network and understand how it works from previous discussion. The following section is to give a brief overview of areas where neural networks being applied.

Signal Processing
One of the many applications of neural network is to suppress noise on telephone line. The need for adaptive echo cancellation has become more suppressing with the development of transcontinental satellite links for long-distance telephone circuits. The adaptive nature of neural nets is a perfect tool for this job.

Pattern Recognition
One of the best known neural network applications. Many interesting researches (including our project) deal with recognizing and classifying data patterns.

Medicine
The idea behind this application is to train neural network to store a large number of medical records, each of which includes information on symptoms, diagnosis, and treatment for a particular case. After training, the net can be presented with input consisting of a set of symptoms; it will then find the full stored pattern that represents the "best matched" diagnosis and treatment.

Speech Recognition
Human voice is recognized by a trained neural network and translated into text.

Business
Several applications of neural network are being applied in a number of business settings. One application is mortgage risk assessment (Collin, Ghosh, and Scofield, 1988a, 1988b). Based from the previous experience, neural net will be able to evaluate and give consistent and reliable assessment on mortgage application. Another application of neural network (which is one of my favorites) is stock prediction. Although still filled with many risks, a trained neural net may give valuable advice based from previous training to stockholders.

Method

A number of different types of perceptrons are described in papers written by Rosenblatt (Rosenblatt, 1962) and by Minsky and Papert (Minsky and Papert, 1988). For the experimental purpose of our project, we have selected and trained a single-layer perceptron to recognize handwritten digits. The training data set we used is obtained from UCI Machine Learning Repository (Courtesy of UCI Machine Learning Repository), which consists 1,934 cases of numeric digits of '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9'. Each training case is in digitized format (see Figure 3) of 0s and 1s at a resolution of 32X32. Based from this input format, we designed the perceptron to accept 1,024 input units (32X32=1,024) and 10 output units (see Figure 4). The weights are fully connected between input units and output units. We chose a modified version of binary step function to be our perceptron's activation function. The output of the perceptron is based from this activation function, given the input y_in:

$$f(y\_in) = \begin{cases} 1 \text{ if } y\_in > 0 \\ 0 \text{ if } y\_in = 0 \\ -1 \text{ if } y\_in < 0 \end{cases}$$

```
00000000000000011100000000000000
00000000000011111111000000000000
00000000001111111111110000000000
00000000011111111111111000000000
00000001111111111111111100000000
00000001111111111111111100000000
00000011111111100011111100000000
00000001111100000011111100000000
00000001110000000011111100000000
00000000000000000011111100000000
00000000000000000111111000000000
00000000000000001111111000000000
00000000000000011111110000000000
00000000000000011111110000000000
00000000000001111111000000000000
00000000000011111111000000000000
00000000000011111111000000000000
00000000000111111111110000000000
00000000000111111111111000000000
00000000000011111111111110000000
00000000000001111111111110000000
00000000000000000001111110000000
00000000000000000000111111000000
00000000000000000000111111000000
00000000000000000000111111000000
00000000000000000001111111000000
00000000000000001111111110000000
00000000001111111111111100000000
00000000001111111111111100000000
00000000001111111111000000000000
00000000000111111111000000000000
00000000000000011100000000000000
```

Figure 3 Digitized numeric digit ‘3’

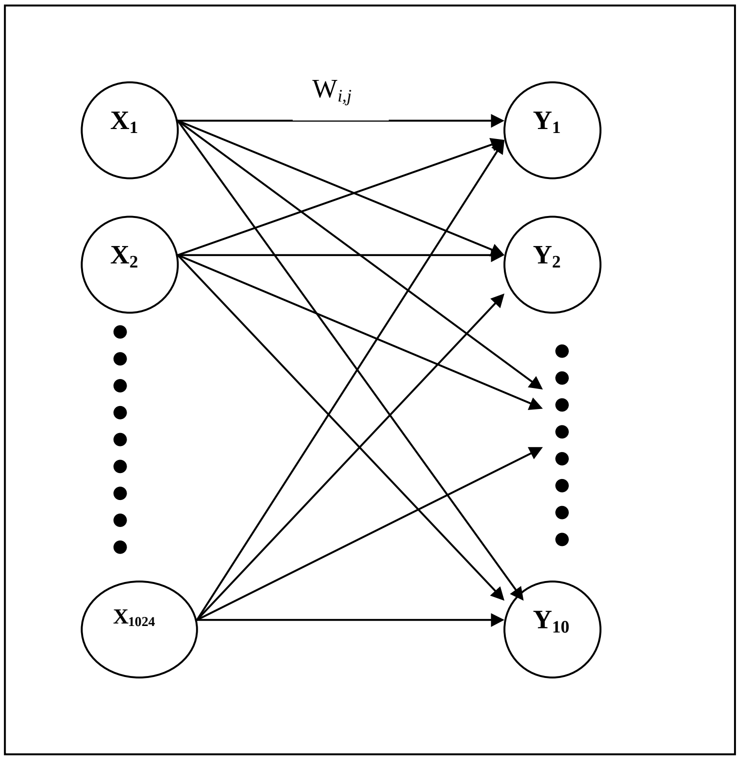

Figure 4 Perceptron with 1024 input units, 10 output units, and full-connected weights

The perceptron algorithm we used in this project is not particularly sensitive to the initial values of weights or the value of the learning rate. For simplicity, the default weights and bias (*b*) are set to 0, and learning rate (α) is set to 1. The following is the modified version of perceptron algorithm based on our initial settings:

$w_{i,j}$	- weight associated with input unit *i* and output unit *j*
s_i	- raw data for input unit *i*
x_i	- input unit *I*
t_j	- true pattern
y_in	- summation of weighted inputs
y	- perceptron output computed by activation function
α	- learning rate, set to 1
b	- bias, set to 0

```
initial weights and bias
do {
        $x_i = s_i$
        $y\_in = \Sigma\, x_i w_{ij}$

        $y = \begin{cases} 1 & \text{if } y\_in > 0 \\ 0 & \text{if } y\_in = 0 \\ -1 & \text{if } y\_in < 0 \end{cases}$
        if $y \neq t$ then      // update weights
            $w_{i,j}$ (new) = $w_{i,j}$ (old) + $t_j x_i$
        else                    // no change
            $w_{i,j}$ (new) = $w_{i,j}$ (old)
    } while (at least one weight has been updated)
```

We also modified the training set which we obtained from UCI Machine Learning Repository, all parameters (except number of training patterns) in the beginning of the file were removed.

Evaluation & Discussion

We tested our perceptron neural network on the handwritten digits data set in the UCI machine learning repository. Given a training data set and a separate testing data set, we trained the perceptron neural network using the training set and test it against the testing set. The training is done in multiple passes. During a pass, each data point in the training set is fed into the perceptron neural network. If the output is different from the target label, then the weights of the perceptron neural network are changed using the perceptron learning rule, otherwise the weights remain the same. After we run a pass through the training set, we test the perceptron neural network against the testing set and record the error rate. One question arises is when to stop the training. We give a plot of training and testing error rate for 50 passes(See Figure 5). Here we use heuristic based on the observation that when a learning algorithm starts overfitting, the error rate on the testing set will go up. Our heuristic is to run for 5 passes so that the algorithm will stabilize, and then monitor the testing error rate; if the testing error rate starts going up, we stop. So we use the testing error rate at point A.

To get an unbiased estimate of the prediction error rate of the algorithm, we use ten-fold cross-validation to evaluate the prediction error rate. The whole data set is divided into 10 bins. (More generally, the data set can be divided into k bins, in which case we have the k-fold cross validation.) Each bin will be used as the testing set in turn when the remaining 9 bins are used as the training set. The prediction error rates are averaged to give an estimate of the prediction error rate on the algorithm for future unseen data. In the following, we report the results in terms of cross validated testing error rate.

There are two formats of the data set. The first one is the original 32x32 bitmap format. In the second data format, the 32x32 bitmap is divided into non-overlapping blocks of 4x4 and the number of "ON" pixels is counted. It is very interesting to compare the relative advantages and disadvantages of the two formats. The preprocessed 8x8 blocks format reduces the dimensions from 1024 (32x32) to 64 (8x8). This will speed up computation. However, it is debatable whether it can improve accuracy; on the one hand, it gives invariance to small distortions; but on the other hand, it loses some original information.

Here are the results: for the 32x32 bitmap format, the error rate is 9.98%; for 8x8 blocks format, the error rate is 11.07%. We are not sure whether this difference of 1% is significant; if yes, we can say the 8x8 blocks format really loses much of the information in the original 32x32 bitmap format.

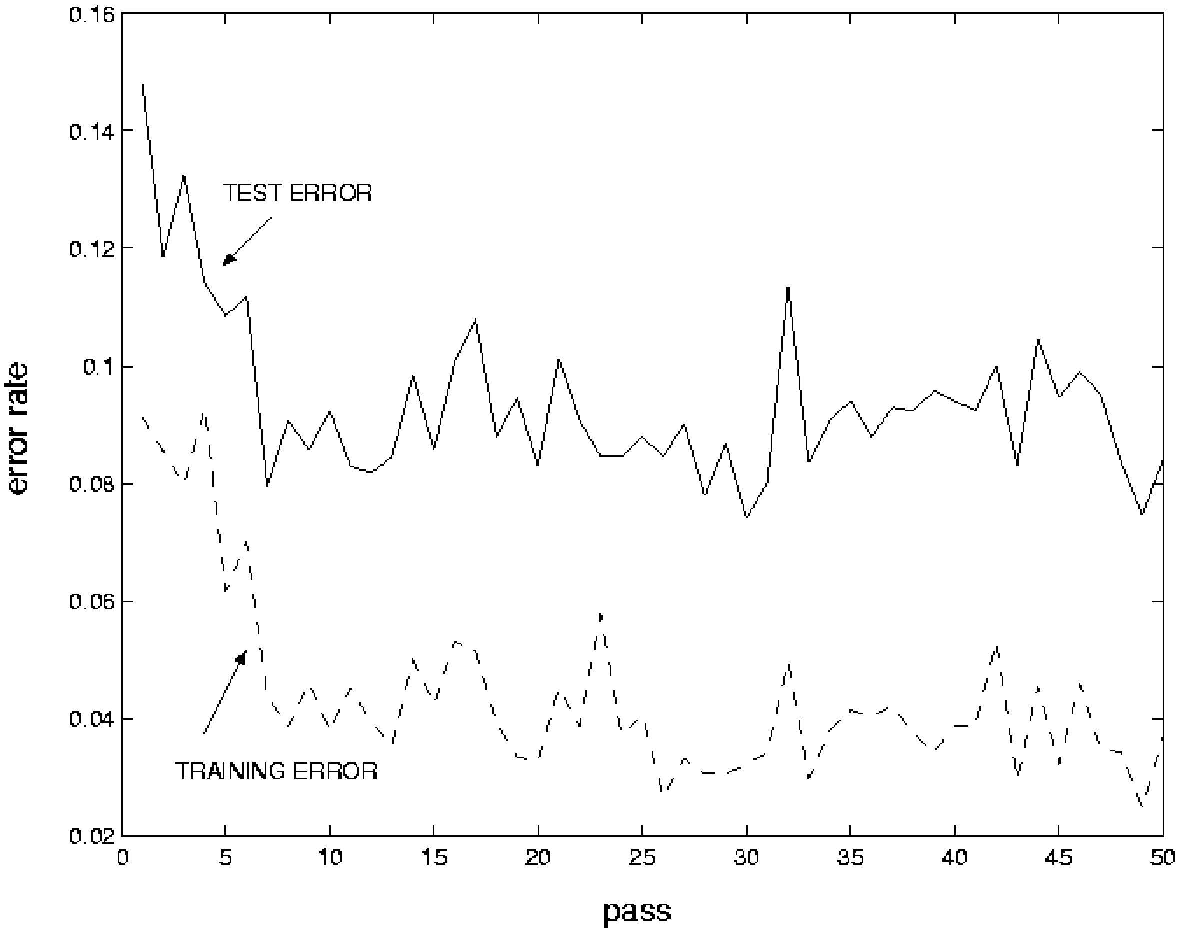

Figure 1 Training and testing error rates for 50 passes

imply that the W0 term is an unnecessary degree of freedom; zero threshold is adequate for this data set of handwritten digits.

We also look at some variations on the algorithm. For example, in our basic algorithm, we do not include the W0 term in the weight vector, which in effect fix the threshold to zero. We had expected to get a lower error rate when we include the W0 term for non-zero threshold. But it turns out that this does not help. In fact, the error rate on the 32x32 bitmap format increases slightly from 9.98% to 10.69%, and from 11.07% to 12.10% on the 8x8 blocks format. This may

We were also curious about the capability of perceptron. We believed, based from the design of perceptron architecture, that it will recognize other patterns as well, provided with sufficient amount of training set. As an extension to our project, we have created a small training set at a lower resolution (6X6) with three distinct patterns: A, F, Chinese character for “day” (see Figure 6). As we have expected, the perceptron recognizes the patterns once it has been properly trained.

```
000100
001010
010010
011111
010001
100001
```

Figure 6 Character ‘A’

Conclusion

In this project, we have successfully applied perceptron learning algorithm to the problem of handwritten digits recognition. Our cross-validated error rate is 9.98%. We find the algorithm runs best (in terms of lowest error rate) when setting threshold to zero and training directly on the original 32x32 bitmap format. Allowing non-zero threshold does not improve perceptron learning, nor does dimension reductions using the 8x8 blocks format, although the latter speed up computation. Our perceptron model can recognize other Character data sets besides handwritten Digits.

Acknowledgement

We would like to thank Professor John Gennari for his helpful suggestions and the pointer to the data set we used in this project.

References

[1] McCulloch, W.S. and Pitts, W., “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[2] Gibson, G.J. and Cowan C.F.N., “On the decision regions of multilayer perceptrons,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1590-1594, October 1990.

[3] Murphy, O.J., “Nearest neighbor pattern classification perceptrons,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1595-1598, October 1990.

[4] Shynk, J.J., “Convergence properties and stationary points of a perceptron learning algorithm,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1599-1604, October 1990.

[5] Minsky, M.L. and Papert, S.A., *Perceptron, expanded edition*, Cambridge, Massachusetts: The MIT Press, 1988.

[6] Collins, E., Ghosh, S., and Scofield, C.L., “An application of a multiple neural network learning system to emulation of mortgage underwriting judgements,” *IEEE International Conference on Neural Network*, pp. 459-466, 1988.

[7] Collins, E., Ghosh, S. and Scofield, C.L., “A neural network decision learning system applied to risk analysis: mortgage underwriting and delinquency risk assessment,” In Holtz, ed., *DARPA Neural Network Study: Part IV System Application*, pp. 65-79, 1988.

[8] Rosenblatt, F., *Principles of Neurodynamics*, New York: Spartan, 1962.