Feedforward network (Perceptron Network)

Sungchul Lee

I use perceptron network to recognize the handwritten digit database in the MNIST.

The perceptron network is very simply neurons network like figure 1.

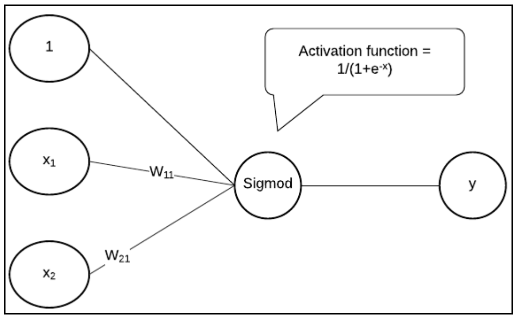


Figure 1. Perceptron network with single hidden neuron

The perceptron network use sigmoid ($1/(1+e^{-x})$) or hyperbolic tangent ($\tanh(x)$) as an activation function. In the homework, I use sigmoid function as an activation function. The network uses '1' constant input for the bias vector in the hidden layer. The detail of perceptron network will be talked with figure 2 and my code. The handwritten image in the MNIST has 28X28 pixels.

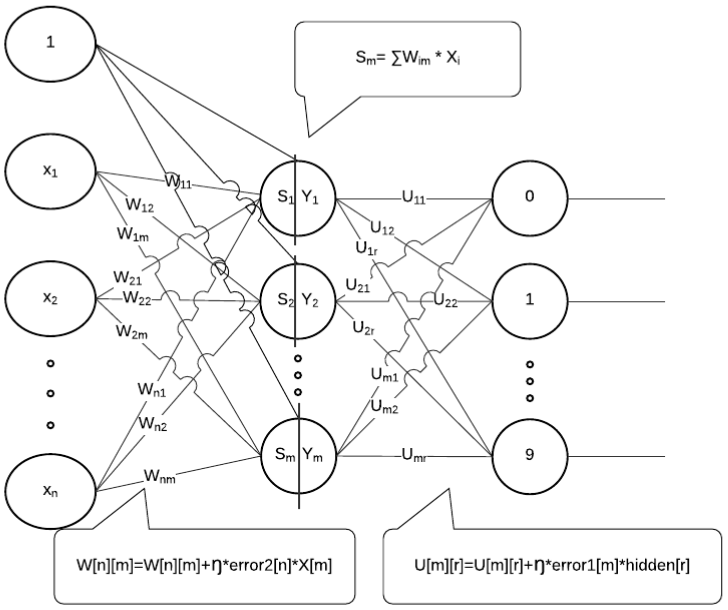


Figure 2. Perceptron network with several hidden neurons

Therefore, I have 784 inputs (X_1 ~ X_{784}) plus 1 which is bias vector. So, there are total 785 inputs per image. Also, the image shows 0 ~ 10. Therefore, we have 10 target outputs. I train the neural network via supervised learning. We already know inputs and outputs to get W and U in figure 2.

SYMBOL	
E	target[i-1] - output[i]
T	target
w	weight
X	input
Z	output
sigmoid()	1/ $(1+e^{-z})$
S	Hidden
Y	Pass through sigmod()

```
-   Code
-   //
-   // E = $\sum_{i=0}^{10}(Z_i - T_i)$
-   //$\frac{dE}{du}$ = ŋ* Z *( 1-Z)(Z-T)* Y
-
-   for (int i = 1; i < 10+1; i++)
-               error2[i] = output[i] * (1.0 - output[i]) * (output[i] - target[i-1]);
-
-   //$\frac{dE}{dw}$ = ŋ* Z *( 1-Z)(Z-T)* X
-            for (int m = 0; m <= hiddenNode; m++)
-            {
-               for (int r = 1; r < 10+1; r++)
-                   sum += weight1[r][m] * error2[r];
-
-               error1[m] = hidden[m] * (1.0 - hidden[m]) * sum;
-               sum = 0.0;
-            }

    //$W_{11(1)}$ = $W_{11(0)}$ - ŋ* Z *( 1-Z)(Z-T)* X

-   //the gradient descent
-            for (int r = 1; r < 10+1; r++)
-               for (int m = 0; m < hiddenNode +1; m++)
-                   W[r][m] -= senstive * error2[r] * hidden[m];
-
-            for (int m = 1; m < hiddenNode; m++)
-               for (int n = 0; n < 28*28 +1; n++)
-                   U[m][n] -= sensitive * error1[m] * input[n];
```

The sensitive is an error rate. The user set the error rate before operating the neural network. The output [] are passed the hidden layer before modifying W and U. The error1[] and error2[] get from different between output[] and target[]. The hidden[] are passed the sigmoid.

//S$_1$ = $\sum_{i=0}^{28*28+1} W_{i1}X_i$

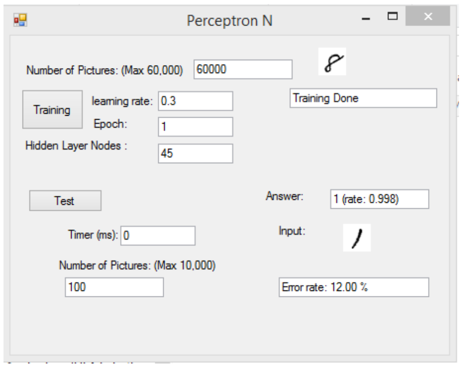
hidden[m] = $\sum_{n=0}^{28*28+1}(W_{mn} * input[n])$

//pass sigmoid

hidden[m] = 1 / (1 + Math.Exp(-hidden[m])

By the code, I can train the neural network.

I can test the test-image in the MNIST via the trained the neural network.


Perceptron N

Number of Pictures: (Max 60,000) 60000

Training

learning rate: 0.3

Epoch: 1

Hidden Layer Nodes : 45

Training Done

Test

Answer: 1 (rate: 0.998)

Timer (ms): 0

Input:

Number of Pictures: (Max 10,000) 100

Error rate: 12.00 %

Figure 3. Perceptron program

Figure 3 show the program of Perceptron, which is made by C#. The optimal size of the hidden layer is usually between the size of the input and size of the output layer. I set the number of hidden layers by following formula [1][2]:

1) The number of hidden neurons should be between the size of the input layer and the size of the output layer.
2) The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
3) The number of hidden neurons should be less than twice the size of the input layer.
4) # of Hidden neuron = $\frac{\#\ of\ sample\ data}{(alpha*(\#\ of\ input\ neuron + \#\ of\ output\ neuron))}$, alpha (2~10)

In my case, the 4th is most suitable than others. 4th shows better accuracy with a small number of Epoch. However, if I train the neuron with enough number of the epoch, the program with another method of hidden neuron shows similar accuracy.

Reference

[1] Jeff Heaton. “Introduction to Neural Networks for Java, 2nd Edition,” ISBN: 1604390085

[2] L. Fletcher, Pretoria, V. Katkovnik, F. E. Steffens and A. P. Engelbrecht,“Optimizing the number of hidden nodes of a feedforward artificial neural network”, Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on (Volume:2). 4-9 May 1998. Anchorage, AK, pp. 1608 - 1612 vol.2. IEEE