MATLAB Deep Learning Notes V

Key Words : Back-Propagation Algorithm

1. Back-Propagation Algorithm

1. As mentioned before, in order to overcome the limitations of the single-layer neural network, we have to use a multi-layer architecture. It is noteworthy that the training process is the only method for the neural network to store information, and untrainable neural networks are useless. We often use **Back propagation algorithm** to train multi-layer neural networks.

2. As shown blew, consider a neural network that consists of two nodes for both the input and output and a hidden layer, which has two nodes as well, and the bias is omitted.

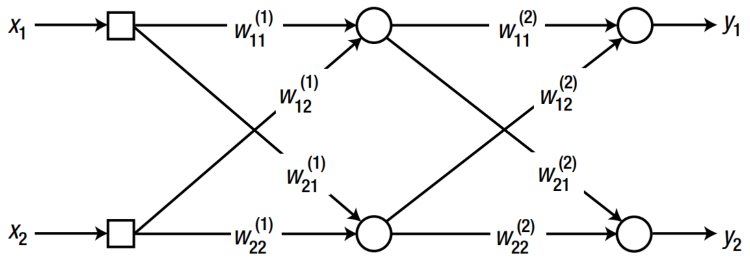

Figure 1: Neural network that consists of two nodes for the input and output and a hidden layer, which has two nodes.

3. In order to obtain the output error, we first need the neural network's output from the input data. It is logical that

$$\begin{bmatrix} v_1^{(1)} \\ v_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \triangleq \mathbf{W}_1 \mathbf{x} \tag{12}$$

4. When we put this weighted sum, Equation 12, into the activation function, we obtain the output from the hidden nodes as

$$\begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} = \begin{bmatrix} \varphi\big(v_1^{(1)}\big) \\ \varphi\big(v_2^{(1)}\big) \end{bmatrix} \tag{13}$$

5. In a similar manner, the weighted sum of the output nodes is calculated as

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} \triangleq \mathbf{W}_2 \mathbf{y}^{(\mathbf{1})} \tag{14}$$

and then

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \varphi\big(v_1\big) \\ \varphi\big(v_2\big) \end{bmatrix} \tag{15}$$

so far we have already finished the forward propagation. Next, we use **Back-propagation algorithm** to train this neural network.

6. First, we calculate the delta of the output nodes.

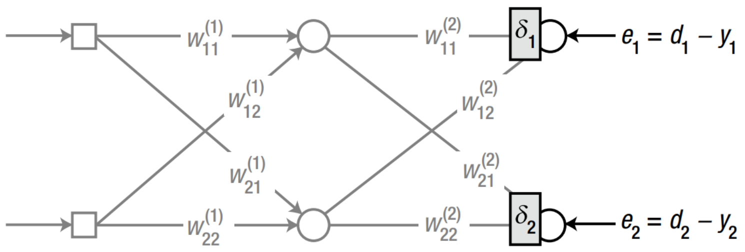

Figure 2: The delta of the output node is defined identically to the delta rule we talked before.

so it is natural to obtain (16) as follow equations

$$\begin{aligned} e_1 &= d_1 - y_1 \\ \delta_1 &= \varphi'(v_1)e_1 \\ e_2 &= d_2 - y_2 \\ \delta_2 &= \varphi'(v_2)e_2 \end{aligned}$$

where $\delta'(\cdot)$ refers to the derivative of the activation function of the output node, y_i is the output from the output node, d_i denotes the correct output from training data, and v_i is the weighted sum of the corresponding node.

7. The most important thing of Back-propagation algorithm is how to define the error of the hidden nodes.

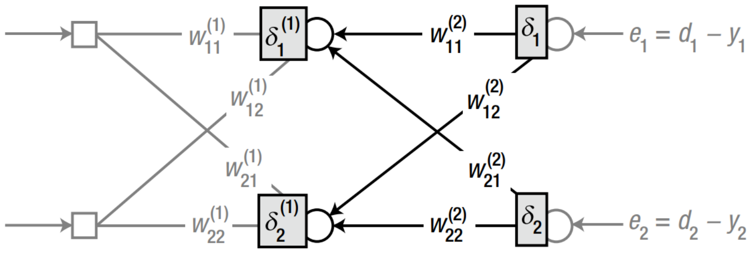

Figure 3: Proceed leftward to the hidden nodes and calculate the delta.

8. From the figure above we have (17)

$$\begin{aligned}
e_1^{(1)} &= w_{11}^{(2)}\delta_1 + w_{21}^{(2)}\delta_2 \\
\delta_1^{(1)} &= \varphi'(v_1^{(1)})e_1^{(1)} \\
e_2^{(1)} &= w_{12}^{(2)}\delta_2 + w_{22}^{(2)}\delta_2 \\
\delta_2^{(1)} &= \varphi'(v_2^{(1)})e_2^{(1)}
\end{aligned}$$

9. In summary, the error of the hidden node is calculated as the backward weighted sum of the delta, and the delta of the node is the product of the error and the derivative of the activation function. This process begins at the output layer and repeats for all hidden layers. We can rewrite (17) as follow

$$\begin{bmatrix} e_1^{(1)} \\ e_2^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \triangleq \mathbf{W}_2^T \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \tag{18}$$

10. This equation indicates that we can obtain the error as the product of the transposed weight matrix and delta vector. If we have additional hidden layers, we will just repeat the same backward process for each hidden layer and calculate all the deltas. Once all the deltas have been calculated, we will be ready to train the neural network using the following equation(19).

$$\begin{aligned}
\Delta w_{ij} &= \alpha\delta_i x_j \\
w_{ij} &\leftarrow w_{ij} + \Delta w_{ij}
\end{aligned}$$

11. Let's organize the process to train the neural network using Back-propagation algorithm as follow.

Algorithm 1 Back-Propagation Algorithm

```
Input:
    Training Data which is formatted as {input, correct output};
Output:
    The neural network with appropriate weights;
 1: Initialize the weights at adequate values;
 2: for each epoch $i \in [1, \text{the number of epochs}]$ do
 3:    for each training data point $j \in [1, \text{the number of training data points}]$ do
 4:       Calculate the error of the output to the correct output and the delta of the output
          nodes;
 5:       for each hidden layer $k \in [1, \text{the number of hidden layers}]$ do
 6:          Propagate the output node delta backward, and calculate the deltas of the imme-
             diate next (left) nodes;
 7:       end for
 8:       Adjust the weights according to the equation (19)
 9:    end for
10: end for
```

2. The Implement of Back-Propagation Algorithm

12. We try to solve the linear inseparable problem by employing Back-propagation algorithm. First, we design a neural network which has one hidden layer as blew

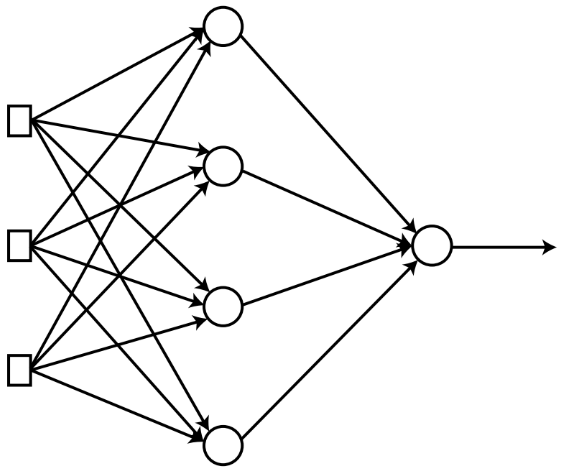

Figure 4: Neural network that consists of three input nodes and a single output node.

Program 10: BackPropXOR

Listing 1: BackPropXOR.m

```
function [weight1, weight2] = BackPropXOR(weight1, weight2, data_input,...
, correct_output)

alpha = 0.9;
N = 4;

for k = 1 : N
    x = data_input(k, :)';
    d = correct_output(k);

    v1 = weight1 * x;
    y1 = sigmoid(v1);
    v = weight2 * y1;
    y = sigmoid(v);

    e = d - y; % calculate the error of the output to the correct output
    delta = y .* (1 - y) .* e;

    e1 = weight2' * delta; % calculate the error of the hidden layer
    delta1 = y1 .* (1 - y1) .* e1;

    % Adjust the weights
    dw1 = alpha * delta1 * x';
    weight1 = weight1 + dw1;

    dw2 = alpha * delta * y1';
    weight2 = weight2 + dw2;
end
end
```

Program 11: testBackPropXOR

Listing 2: testBackPropXOR.m

```
clear
data_input = [ 0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1, 1]; % training data
correct_output = [0; 1; 1; 0]; % correct outputs(i.e., labels)
weight1 = 2 * rand(4, 3) - 1;
weight2 = 2 * rand(1, 4) - 1;


for epoch = 1:10000 % train
```

```
[weight1, weight2] = BackPropXOR(weight1, weight2, data_input,...
, correct_output);
end

N = 4; % inference
for k = 1 : N
x = data_input(k, :)';
v1 = weight1 * x;
y1 = sigmoid(v1);
v = weight2 * y1;
y = sigmoid(v)
end
```

Output 11:

```
>> testBackPropXOR

y =
    0.0079

y =
    0.9905

y =
    0.9905

y =
    0.0136
```

13. So we successfully solved the linear inseparable problem by using Back-propagation algorithm. Interestingly, unlike the single-layer neural network, we would obtain slightly different weights each time we train the network.

$$\begin{bmatrix} 0.0060 \\ 0.9888 \\ 0.9891 \\ 0.0134 \end{bmatrix} \quad \Leftrightarrow \quad D = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figure 5: These values are very close to the correct output.