MATLAB Deep Learning Notes VIII

Key Words : Classification; Softmax Activation Function; One-Hot Encoding

1. Binary Classification

1. For binary classification, the neural network is constructed with a single output node and sigmoid activation function. **The correct output of the training data is converted to the maximum and minimum values** of the activation function. The cost function of the learning rule employs the cross entropy function

2. Softmax Activation Function

2. The softmax activation function is

$$y_i = \varphi(v_i) = \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}$$

3. Binary and Multiclass classifier networks employ different activation functions — the sigmoid for the binary and the softmax for the multiclass. We should clear that the softmax function accounts not only for the weighted sum of the inputs, but also for the inputs to the other output nodes, and this feature is very useful to multiclass classification problems.

Program 18: softmax

Listing 1: softmax.m

```
function [ y ] = softmax( x )
ex = exp( x );
y = ex / sum( ex );
end
```

Output 18:

```
>> a = softmax( [1, 2, 3] )

a =
    0.0900    0.2447    0.6652

>> sum(a)

ans =

     1
```

3. One-Hot Encoding

4. In order to calculate the error, we should switch the class names into numeric codes. Considering that we have three output nodes from the neural network, we create the classes as the following vectors:

$$\text{Class 1} \rightarrow [1, 0, 0]$$
$$\text{Class 2} \rightarrow [0, 1, 0]$$
$$\text{Class 3} \rightarrow [0, 0, 1]$$

5. This transformation implies that each output node is mapped to an element of the class vector, which only yields 1 for the corresponding node. See Fig.1 below

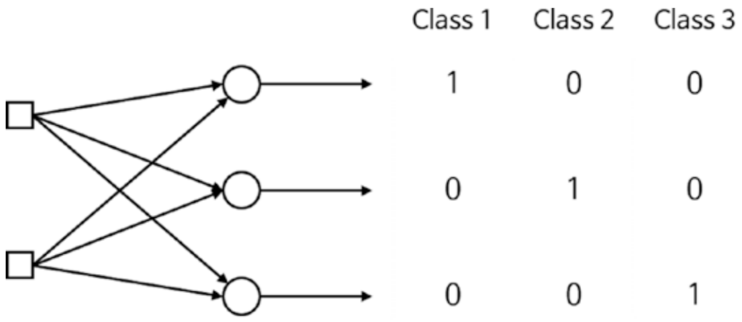

Figure 1: Each output node is now mapped to an element of the class vector.

4. Multiclass Classification

6. We implement a multiclass classifier network that recognizes digits from the input images. This is a multiclass classification, as it classifies the image into specified digits. The input images are five-by-five pixel squares, which display five numbers from 1 to 5, as shown below.

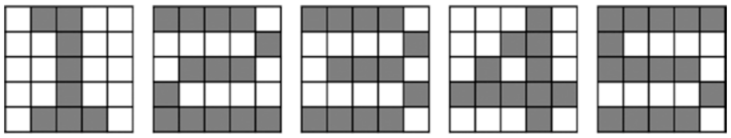

Figure 2: Five-by-five pixel squares that display five numbers from 1 to 5.

7. We can convert Fig.2 into five matrix, which are shown below

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \Rightarrow \mathbf{1}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \Rightarrow \mathbf{2}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \Rightarrow \mathbf{3}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \Rightarrow \mathbf{4}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \Rightarrow \mathbf{5}$$

Program 19: MultiClass

Listing 2: MultiClass.m

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

alpha = 0.9;
N = 5;

for k = 1 : N
    x = reshape(data_input(:, :, k), 25, 1); %size(data_input) = [5, 5, 5]
    d = correct_output(k, :)';

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

    e = d - y; % calculate the error of the output to the correct output
    delta = 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
```

8. Next we test the MultiClass.m

Program 20: testMultiClass

Listing 3: testMultiClass.m

```
clear

X = zeros(5, 5, 5);

X(:, :, 1) = [ 0 1 1 0 0;
               0 0 1 0 0;
               0 0 1 0 0;
               0 0 1 0 0;
               0 1 1 1 0
             ];
X(:, :, 2) = [ 1 1 1 1 0;
               0 0 0 0 1;
               0 1 1 1 0;
               1 0 0 0 0;
               1 1 1 1 1
             ];
X(:, :, 3) = [ 1 1 1 1 0;
               0 0 0 0 1;
               0 1 1 1 0;
               0 0 0 0 1;
               1 1 1 1 0
             ];
X(:, :, 4) = [ 0 0 0 1 0;
               0 0 1 1 0;
               0 1 0 1 0;
               1 1 1 1 1;
               0 0 0 1 0
             ];
X(:, :, 5) = [ 1 1 1 1 1;
               1 0 0 0 0;
               1 1 1 1 0;
               0 0 0 0 1;
               1 1 1 1 0
             ];
D = [ 1 0 0 0 0;
      0 1 0 0 0;
      0 0 1 0 0;
      0 0 0 1 0;
      0 0 0 0 1
    ];
weight1 = 2*rand(50, 25) - 1;
weight2 = 2*rand( 5, 50) - 1;

for epoch = 1:10000 % train
    [weight1, weight2] = MultiClass(weight1, weight2, X, D);
end
N = 5; % inference
for k = 1:N
    x = reshape(X(:, :, k), 25, 1);
    v1 = weight1 * x;
    y1 = sigmoid(v1);
    v = weight2 * y1;
    y = softmax(v)
end
```

Output 20:

```
>> testMultiClass

y =

    1.0000
    0.0000
    0.0000
    0.0000
    0.0000

y =

    0.0000
    1.0000
    0.0000
    0.0000
    0.0000

y =...
```

9. It seems perfect. Next we use real data to verify.

   Program 21: testRealMultiClass

Listing 4: testRealMultiClass.m

```
clear

testMultiClass; % train weight1, weight2

X = zeros(5, 5, 5);

X(:, :, 1) = [ 0 0 1 1 0;
               0 0 1 1 0;
               0 1 0 1 0;
               0 0 0 1 0;
               0 1 1 1 0
              ];
```

```
X(:, :, 2) = [ 1 1 1 1 0;
               0 0 0 0 1;
               0 1 1 1 0;
               1 0 0 0 1;
               1 1 1 1 1
              ];
X(:, :, 3) = [ 1 1 1 1 0;
               0 0 0 0 1;
               0 1 1 1 0;
               1 0 0 0 1;
               1 1 1 1 0
              ];
X(:, :, 4) = [ 0 1 1 1 0;
               0 1 0 0 0;
               0 1 1 1 0;
               0 0 0 1 0;
               0 1 1 1 0
              ];
X(:, :, 5) = [ 0 1 1 1 1;
               0 1 0 0 0;
               0 1 1 1 0;
               0 0 0 1 0;
               1 1 1 1 0
              ];

N = 5; % inference
for k = 1:N
x = reshape(X(:, :, k), 25, 1);
v1 = weight1 * x;
y1 = sigmoid(v1);
v = weight2 * y1;
y = softmax(v)
end
```

Output 21:

```
y =

    0.0172
    0.0975
    0.0000
    0.7114
    0.1739
```

```
y =

    0.0000
    0.9946
    0.0050
    0.0001
    0.0003

y =

    0.0001
    0.0584
    0.9392
    0.0007
    0.0016

y =

    0.1739
    0.5557
    0.1099
    0.0081
    0.1524

y =

    0.0135
    0.5327
    0.0115
    0.0069
    0.4355
```

10. The result looks good!Let's look at the fourth image. It is classified as a 5 by 47.12% probability. At the same time, it could be a 3 by a pretty high probability of 32.08%. Let's see what happened. The input image appears to be a squeezed 5. Furthermore, the neural network finds some horizontal lines that resemble features of a 3, therefore giving that a high probability. In this case, the neural network should be trained to have more variety in the training data in order to improve its performance