Neural Network and Classification

- Binary Classification
- Multiclass Classification
- Example: Multiclass Classification

Binary Classification

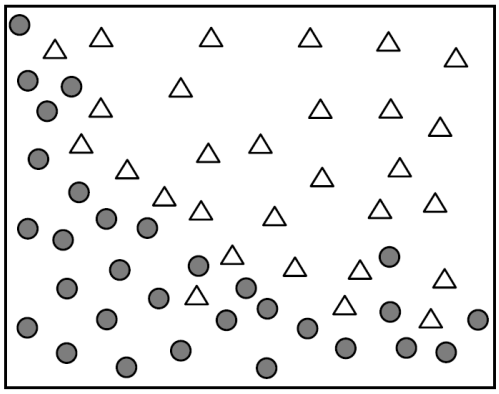


Figure 4-1. *Binary classification problem*

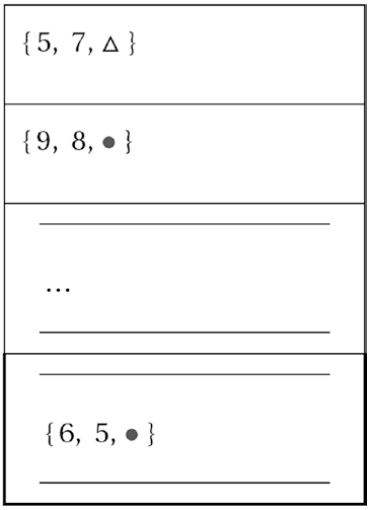


Figure 4-2. *Training data binary classification*

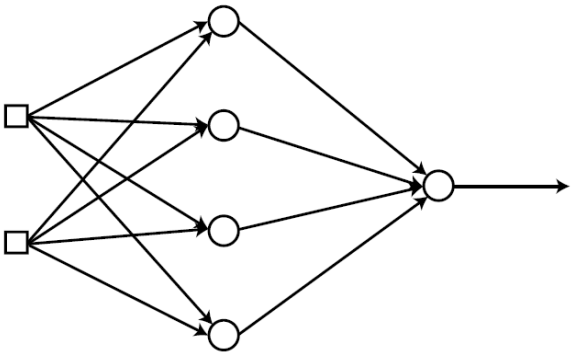


Figure 4-3. *Neural network for the training data*

The neural network produces numerical outputs that range from 0-1, while we have the

symbolic correct outputs given as △ and ●. We cannot calculate the error in this way; we need to switch the symbols to numerical codes. We can assign the maximum and minimum values of the sigmoid function to the two classes as follows:

Class △ → 1

Class ● → 0

{5, 7, **1**}
{9, 8, **0**}
…
{6, 5, **0**}

Figure 4-4*. Change the class symbols and the data is classified differently*

Multiclass Classification

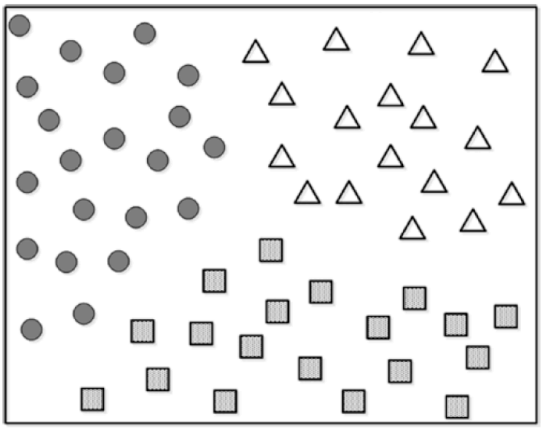


Figure 4-5. *Data with three classes*

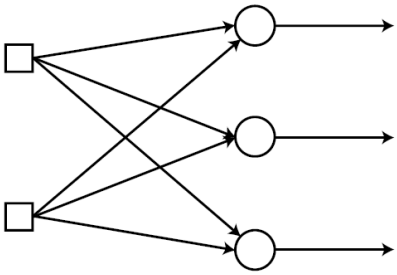


Figure 4-6. *Configured neural network for the three classes*

{5, 7, Class1 }
{9, 8, Class 3 }
{2, 4, Class 2 }
…
{6, 5, Class 3 }

Figure 4-7. *Training data with multiclass classifier*

In order to calculate the error, we switch the class names into numeric codes, as we did in the previous section.

Class 1 → [1 0 0]
Class 2 → [0 1 0]
Class 3 → [0 0 1]

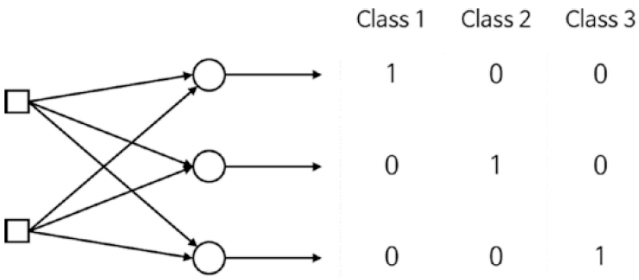


Figure 4-8. *Each output node is now mapped to an element of the class vector*

This expression technique is called one-hot encoding or 1-of-N encoding.

Now, the training data is displayed in the following format:

{5, 7, 1, 0, 0}
{9, 8, 0, 0, 1}
{2, 4, 0, 1, 0}
…
{6, 5, 0, 0, 1}

Figure 4-9. *Training data is displayed in a new format*

$$v = \begin{bmatrix} 2 \\ 1 \\ 0.1 \end{bmatrix} \quad \Rightarrow \quad \varphi(v) = \begin{bmatrix} \dfrac{e^2}{e^2 + e^1 + e^{0.1}} \\ \dfrac{e^1}{e^2 + e^1 + e^{0.1}} \\ \dfrac{e^{0.1}}{e^2 + e^1 + e^{0.1}} \end{bmatrix} = \begin{bmatrix} 0.6590 \\ 0.2424 \\ 0.0986 \end{bmatrix}$$

Figure 4-10. *Softmax function calculations*

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Figure 4-11. *Output when using a sigmoid function*

The output from the i-th output node of the softmax function is calculated as follows:

$$y_i = \varphi(v_i) = \frac{e^{v_i}}{e^{v_1} + e^{v_2} + e^{v_3} + \cdots + e^{v_M}} = \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}$$

Following this definition, the softmax function satisfies the following condition:

$$\varphi(v_1) + \varphi(v_2) + \varphi(v_3) + \cdots + \varphi(v_M) = 1$$

The derivative of softmax function:

$$\frac{\partial y_i}{\partial v_j} = \begin{cases} \varphi(v_j)(1-\varphi(v_j)) = y_j(1-y_j), i = j \\ -\varphi(v_i)\varphi(v_j) = -y_i y_j, \qquad i \neq j \end{cases}$$

The back propagation of cross entropy function for multiclass classification.

$$loss = -\sum_{i=1}^{C} d_i \ln y_i, \;\; y_i = \varphi(v_i)$$

C-- the number of class

$$\begin{aligned} \frac{\partial loss}{\partial v_j} &= -\sum_{i=1}^{C} \frac{d_i}{y_i}\frac{\partial y_i}{\partial v_j} \\ &= -\sum_{i=j} \frac{d_j}{y_j} y_j(1-y_j) - \sum_{i \neq j} \frac{d_i}{y_i}(-y_i y_j) \\ &= -d_j + y_j \sum_{i=1}^{C} d_i = y_j - d_j \end{aligned}$$

The vector form

$$\frac{\partial loss}{\partial \mathbf{v}} = \mathbf{y} - \mathbf{d}$$

The training process of the multiclass classification neural network is summarized in these steps.

1. Construct the output nodes to have the same value as the number of classes. The softmax function is used as the activation function.

2. Switch the names of the classes into numeric vectors via the one-hot encoding method.

Class 1 --> [1 0 0]

Class 2 --> [0 1 0]

Class 3 --> [0 0 1]

3. Initialize the weights of the neural network with adequate values.

4. Enter the input from the training data { input, correct output } into the neural network and obtain the output. Calculate the error between the output and correct output and determine the delta, δ, of the output nodes.

$$\begin{aligned} e &= d - y \\ \delta &= e \end{aligned}$$

5. Propagate the output delta backwards and calculate the delta of the subsequent hidden nodes.

$$e^{(k)} = W^T \delta$$
$$\delta^{(k)} = \varphi'\left(v^{(k)}\right) e^{(k)}$$

6. Repeat Step 5 until it reaches the hidden layer on the immediate right of the input layer.

7. Adjust the weights of the neural network using this learning rule:

$$\Delta w_{ij} = \alpha \delta_i x_j$$
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

8. Repeat Steps 4-7 for all the training data points.

9. Repeat Steps 4-8 until the neural network has been trained properly.

Example: Multiclass Classification

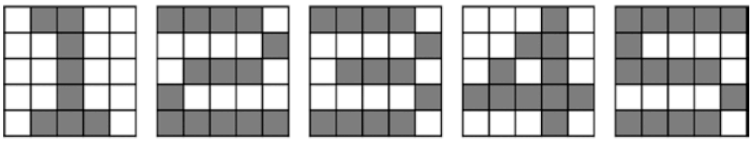


Figure 4-12. *Five-by-five pixel squares that display five numbers from 1 to 5*

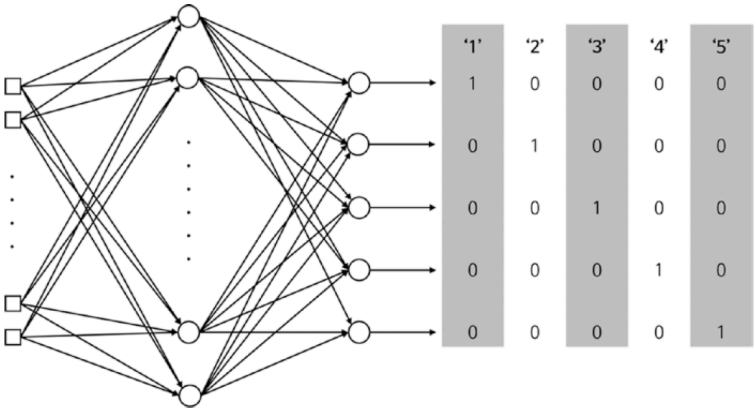


Figure 4-13. *The neural network model for this new dataset*

```
rng(3);

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
    ];

W1 = 2*rand(50, 25) - 1;
W2 = 2*rand( 5, 50) - 1;

for epoch = 1:10000           % train
  [W1,W2] = MultiClass(W1, W2, X, D);
end

N = 5;                        % inference
for k = 1:N
  x  = reshape(X(:, :, k), 25, 1);
  v1 = W1*x;
  y1 = Sigmoid(v1);
  v  = W2*y1;
  y  = Softmax(v)
end
```

```
y = 5×1
    1.0000
    0.0000
    0.0000
    0.0000
    0.0000
y = 5×1
    0.0000
    1.0000
    0.0000
    0.0000
    0.0000
```

```
y = 5×1
    0.0000
    0.0000
    1.0000
    0.0000
    0.0000
y = 5×1
    0.0000
    0.0000
    0.0000
    1.0000
    0.0000
y = 5×1
    0.0000
    0.0000
    0.0000
    0.0000
    1.0000
```

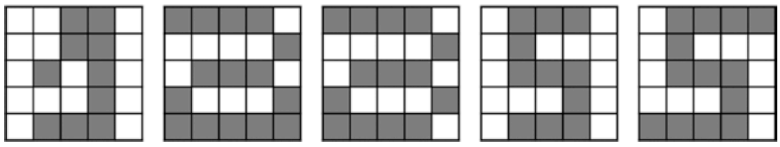
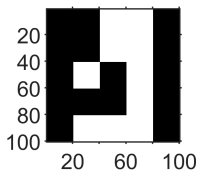


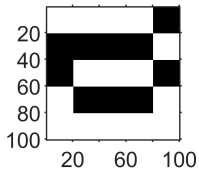
Figure 4-15. *Let's see how the neural network responds to these contaminated images*

```
%TestMultiClass;                % W1, W2

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
             ];
```

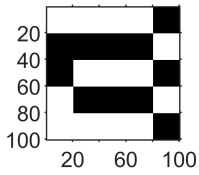
```
X(:, :, 5) = [ 0 1 1 1 1;
               0 1 0 0 0;
               0 1 1 1 0;
               0 0 0 1 0;
               1 1 1 1 0
             ];

N = 5;                          % inference
for k = 1:N
  x  = reshape(X(:, :, k), 25, 1);
  v1 = W1*x;
  y1 = Sigmoid(v1);
  v  = W2*y1;
  figure,imshow(imresize(X(:, :, k),20,'nearest'));axis on
  y  = Softmax(v)
end
```



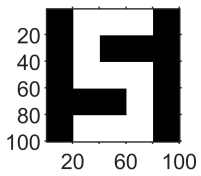

```
y = 5×1
    0.0208
    0.0006
    0.0363
    0.9164
    0.0259
```



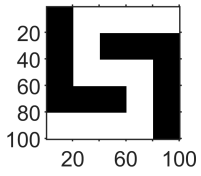

```
y = 5×1
    0.0000
    0.9961
    0.0038
    0.0000
    0.0000
```



```
y = 5×1
    0.0001
    0.0198
    0.9798
    0.0001
    0.0002
```




```
y = 5×1
    0.0930
    0.3057
    0.5397
    0.0408
    0.0208
```




```
y = 5×1
    0.0363
    0.3214
    0.0717
    0.0199
    0.5506
```

Exercise: Classify the iris dataset and compare your result with KNN.

Referenece: KNNClassifier_iris.mlx