CHAPTER 4

Neural Network and Classification

As addressed in Chapter 1, the primary Machine Learning applications that require supervised learning are classification and regression. Classification is used to determine the group the data belongs. Some typical applications of classification are spam mail filtering and character recognition. In contrast, regression infers values from the data. It can be exemplified with the prediction of income for a given age and education level.

Although the neural network is applicable to both classification and regression, it is seldom used for regression. This is not because it yields poor performance, but because most of regression problems can be solved using simpler models. Therefore, we will stick to classification throughout this book.

In the application of the neural network to classification, the output layer is usually formulated differently depending on how many groups the data should be divided into. The selection of the number of nodes and suitable activation functions for the classification of two groups is different when using more groups. Keep in mind that it affects only the output nodes, while the hidden nodes remain intact. Of course, the approaches of this chapter are not only ones available. However, these may be the best to start with, as they have been validated through many studies and cases.

Binary Classification

We will start with the binary classification neural network, which classifies the input data into one of the two groups. This kind of classifier is actually useful for more applications than you may expect. Some typical applications include spam mail filtering (a spam mail or a normal mail) and loan approvals (approve or deny).

P. Kim, *MATLAB Deep Learning*, DOI 10.1007/978-1-4842-2845-6_4

For binary classification, a single output node is sufficient for the neural network. This is because the input data can be classified by the output value, which is either greater than or less than the threshold. For example, if the sigmoid function is employed as the activation function of the output node, the data can be classified by whether the output is greater than 0.5 or not. As the sigmoid function ranges from 0-1, we can divide groups in the middle, as shown in Figure 4-1.

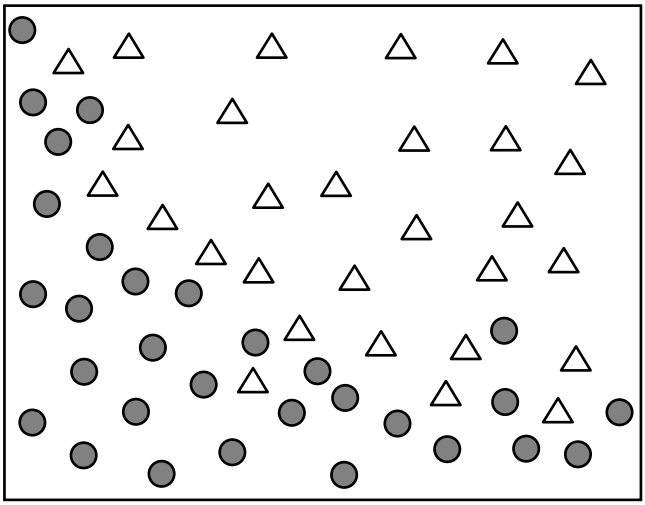

Figure 4-1. *Binary classification problem*

Consider the binary classification problem shown in Figure 4-1. For the given coordinates (x, y), the model is to determine which group the data belongs. In this case, the training data is given in the format shown in Figure 4-2. The first two numbers indicate the x and y coordinates respectively, and the symbol represents the group in which the data belongs. The data consists of the input and correct output as it is used for supervised learning.

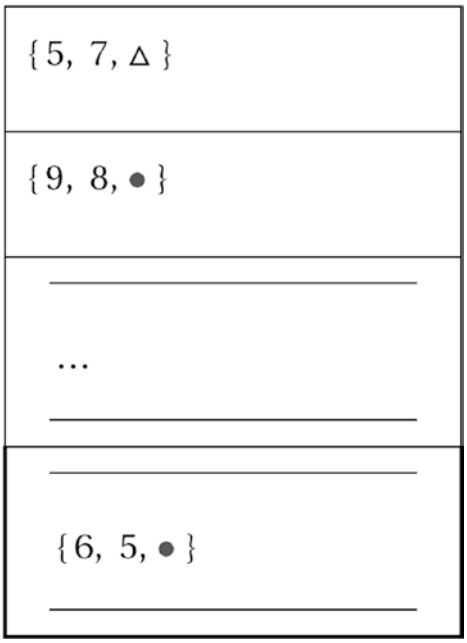

Figure 4-2. *Training data binary classification*

Now, let's construct the neural network. The number of input nodes equals the number of input parameters. As the input of this example consists of two parameters, the network employs two input nodes. We need one output node because this implements the classification of two groups as previously addressed. The sigmoid function is used as the activation function, and the hidden layer has four nodes.[1] Figure 4-3 shows the described neural network.

[1]The hidden layer is not our concern. The layer that varies depending on the number of classes is the output layer, not the hidden layer. There is no standard rule for the composition of the hidden layer.

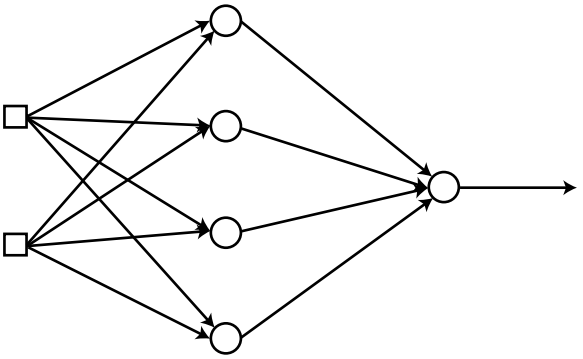

Figure 4-3. *Neural network for the training data*

When we train this network with the given training data, we can get the binary classification that we want. However, there is a problem. 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

The change of the class symbols yields the training data shown in Figure 4-4.

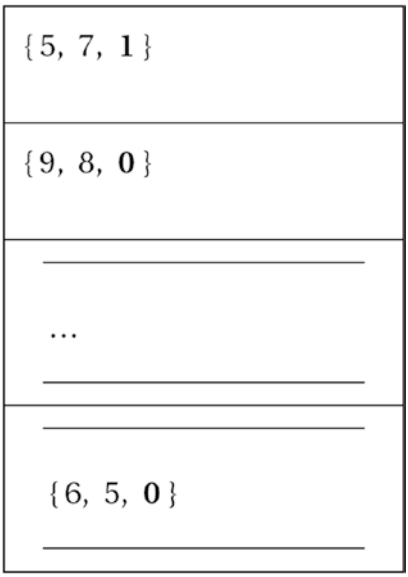

Figure 4-4. *Change the class symbols and the data is classified differently*

The training data shown in Figure 4-4 is what we use to train the neural network. The binary classification neural network usually adopts the cross entropy function of the previous equation for training. You don't have to do so all the time, but it is beneficial for the performance and implementation process. The learning process of the binary classification neural network is summarized in the following steps. Of course, we use the cross entropy function as the cost function and the sigmoid function as the activation function of the hidden and output nodes.

1. The binary classification neural network has one node for the output layer. The sigmoid function is used for the activation function.
2. Switch the class titles of the training data into numbers using the maximum and minimum values of the sigmoid function.

   Class △ → 1

   Class ● → 0
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.

$$e = d - y$$
$$d = e$$

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 training data points.

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

Although it appears complicated because of its many steps, this process is basically the same as that of the back-propagation of Chapter 3. The detailed explanations are omitted.

Multiclass Classification

This section introduces how to utilize the neural network to deal with the classification of three or more classes. Consider a classification of the given inputs of coordinates (x, y) into one of three classes (see Figure 4-5).

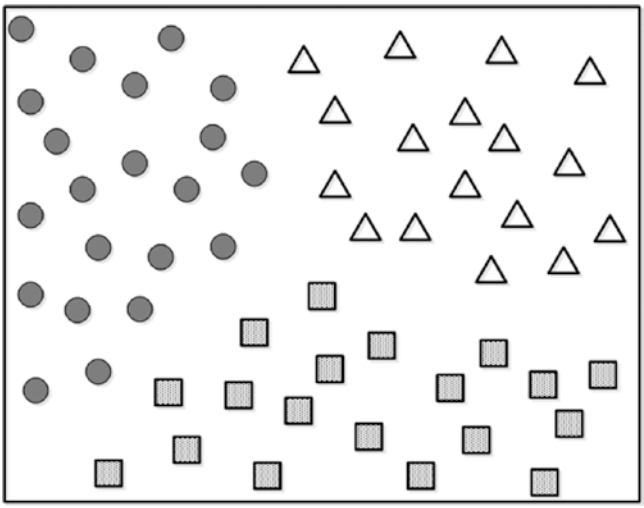

Figure 4-5. *Data with three classes*

We need to construct the neural network first. We will use two nodes for the input layer as the input consists of two parameters. For simplicity, the hidden layers are not considered at this time. We need to determine the number of the output nodes as well. It is widely known that matching the number of output nodes to the number of classes is the most promising method. In this example, we use three output nodes, as the problem requires three classes. Figure 4-6 illustrates the configured neural network.

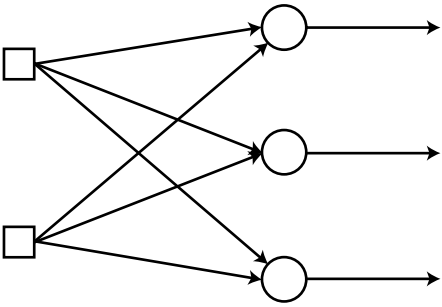

Figure 4-6. *Configured neural network for the three classes*

Once the neural network has been trained with the given data, we obtain the multiclass classifier that we want. The training data is given in Figure 4-7. For each data point, the first two numbers are the x and y coordinates respectively,

and the third value is the corresponding class. The data includes the input and correct output as it is used for supervised learning.

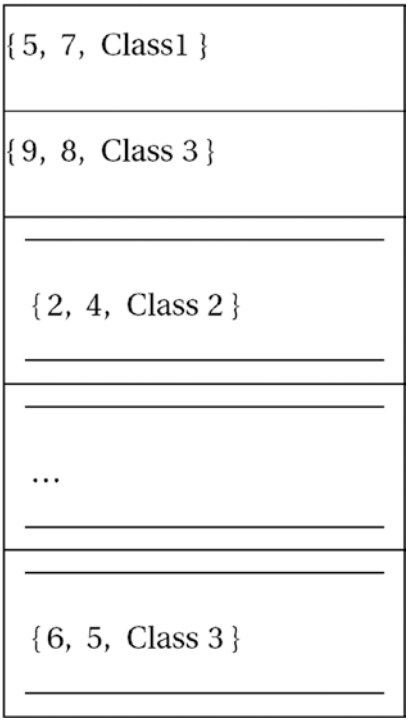

Figure 4-7. *Training data with multiclass classifier*

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

Class 1 → [1 0 0]

Class 2 → [0 1 0]

Class 3 → [0 0 1]

This transformation implies that each output node is mapped to an element of the class vector, which only yields 1 for the corresponding node. For example, if the data belongs to Class 2, the output only yields 1 for the second node and 0 for the others (see Figure 4-8).

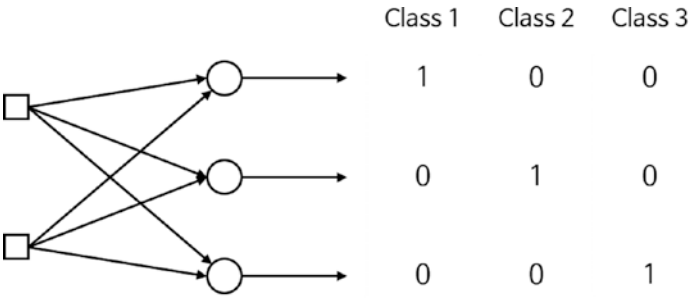

Figure 4-8. *Each output node is now mapped to an element of the class vector*

This expression technique is called *one-hot encoding* or *1-of-N encoding*. The reason that we match the number of output nodes to the number of classes is to apply this encoding technique. Now, the training data is displayed in the format shown in Figure 4-9.

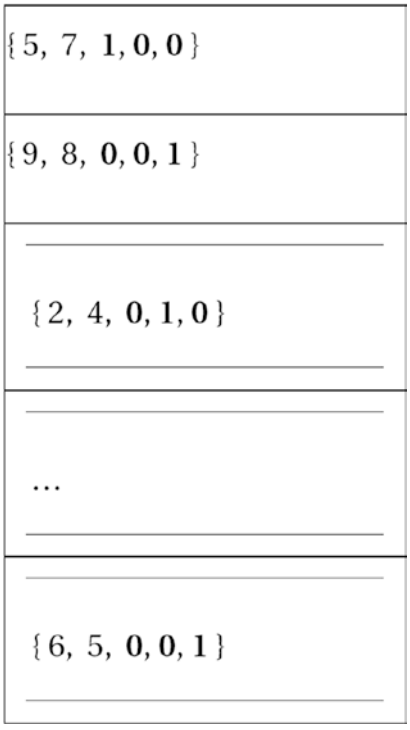

Figure 4-9. *Training data is displayed in a new format*

Next, the activation function of the output node should be defined. Since the correct outputs of the transformed training data range from zero to one, can we just use the sigmoid function as we did for the binary classification? In general, multiclass classifiers employ the softmax function as the activation function of the output node.

The activation functions that we have discussed so far, including the sigmoid function, account only for the weighted sum of inputs. They do not consider the output from the other output nodes. However, the softmax function accounts not only for the weighted sum of the inputs, but also for the inputs to the other output nodes. For example, when the weighted sum of the inputs for the three output nodes are 2, 1, and 0.1, respectively, the softmax function calculates the outputs shown in Figure 4-10. All of the weighted sums of the inputs are required in the denominator.

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

Figure 4-10. *Softmax function calculations*

Why do we insist on using the softmax function? Consider the sigmoid function in place of the softmax function. Assume that the neural network produced the output shown in Figure 4-11 when given the input data. As the sigmoid function concerns only its own output, the output here will be generated.

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

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

The first output node appears to be in Class 1 by 100 percent probability. Does the data belong to Class 1, then? Not so fast. The other output nodes also indicate 100 percent probability of being in Class 2 and Class 3. Therefore, adequate interpretation of the output from the multiclass classification neural network requires consideration of the relative magnitudes of all node outputs. In this example, the actual probability of being each class is $\frac{1}{3}$. The softmax function provides the correct values.

The softmax function maintains the sum of the output values to be one and also limits the individual outputs to be within the values of 0-1. As it accounts for the relative magnitudes of all the outputs, the softmax function is a suitable choice for the multiclass classification neural networks. The output from the *i*-th output node of the softmax function is calculated as follows:

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

where, v_i is the weighted sum of the i-th output node, and M is the number of output nodes. Following this definition, the softmax function satisfies the following condition:

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

Finally, the learning rule should be determined. The multiclass classification neural network usually employs the cross entropy-driven learning rules just like the binary classification network does. This is due to the high learning performance and simplicity that the cross entropy function provides.

Long story short, the learning rule of the multiclass classification neural network is identical to that of the binary classification neural network of the previous section. Although these two neural networks employ different activation functions—the sigmoid for the binary and the softmax for the multiclass—the derivation of the learning rule leads to the same result. Well, it is better for us to have less to remember.

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.

$$e = d - y$$
$$\delta = e$$

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.

Of course, the multiclass classification neural network is applicable for binary classification. All we have to do is construct a neural network with two output nodes and use the softmax function as the activation function.

Example: Multiclass Classification

In this section, we implement a multiclass classifier network that recognizes digits from the input images. The binary classification has been implemented in Chapter 3, where the input coordinates were divided into two groups. As it classified the data into either 0 or 1, it was binary classification.

Consider an image recognition of digits. 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 in Figure 4-12.

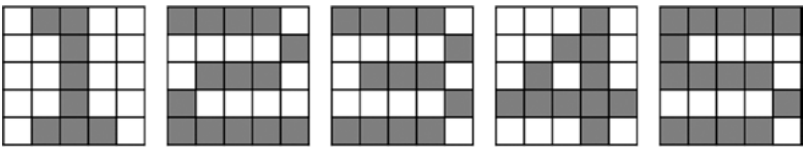

Figure 4-12. *Five-by-five pixel squares that display five numbers from 1 to 5*

The neural network model contains a single hidden layer, as shown in Figure 4-13. As each image is set on a matrix, we set 25 input nodes. In addition, as we have five digits to classify, the network contains five output nodes. The softmax function is used as the activation function of the output node. The hidden layer has 50 nodes and the sigmoid function is used as the activation function.

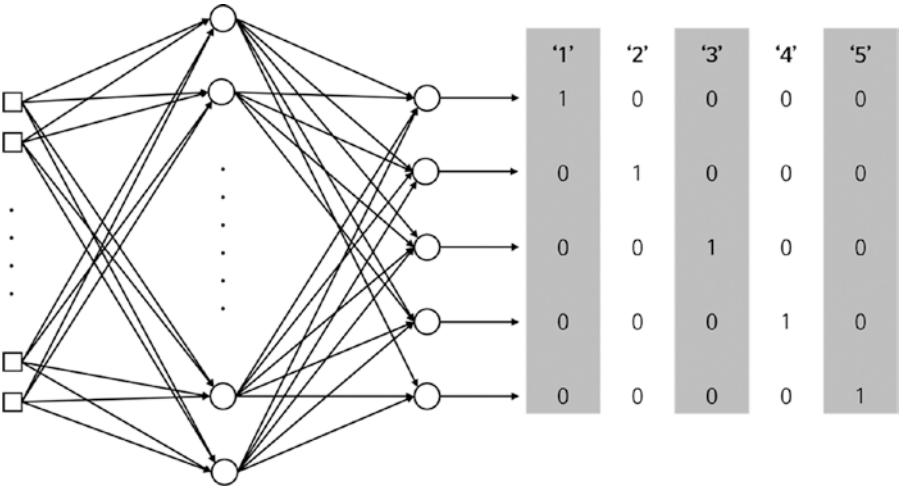

Figure 4-13. *The neural network model for this new dataset*

The function MultiClass implements the learning rule of multiclass classification using the SGD method. It takes the input arguments of the weights and training data and returns the trained weights.

```
[W1, W2] = MultiClass(W1, W2, X, D)
```

where W1 and W2 are the weight matrices of the input-hidden and hidden-output layers, respectively. X and D are the input and correct output of the training data, respectively. The following listing shows the MultiClass.m file, which implements the function MultiClass.

```
function [W1, W2] = MultiClass(W1, W2, X, D)
  alpha = 0.9;

  N = 5;
  for k = 1:N
    x = reshape(X(:, :, k), 25, 1);
    d = D(k, :)';

    v1 = W1*x;
    y1 = Sigmoid(v1);
    v  = W2*y1;
    y  = Softmax(v);

    e     = d - y;
    delta = e;

    e1     = W2'*delta;
    delta1 = y1.*(1-y1).*e1;

    dW1 = alpha*delta1*x';
    W1 = W1 + dW1;

    dW2 = alpha*delta*y1';
    W2 = W2 + dW2;
  end
end
```

This code follows the same procedure as that of the example code in the "Cross Entropy Function" section in Chapter 3, which applies the delta rule to the training data, calculates the weight updates, dW1 and dW2, and adjusts the neural network's weights. However, this code slightly differs in that it uses the

function `softmax` for the calculation of the output and calls the function `reshape` to import the inputs from the training data.

```
x = reshape(X(:, :, k), 25, 1);
```

The input argument `X` contains the stacked two-dimensional image data. This means that `X` is a $5 \times 5 \times 5$ three-dimensional matrix. Therefore, the first argument of the function `reshape, X(:, :, k)` indicates the 5×5 matrix that contains the **k**-th image data. As this neural network is compatible with only the vector format inputs, the two-dimensional matrix should be transformed into a 25×1 vector. The function `reshape` performs this transformation.

Using the cross entropy-driven learning rule, the delta of the output node is calculated as follows:

```
e     = d - y;
delta = e;
```

Similar to the example from Chapter 3, no other calculation is required. This is because, in the cross entropy-driven learning rule that uses the softmax activation function, the delta and error are identical. Of course, the previous back-propagation algorithm applies to the hidden layer.

```
e1     = W2'*delta;
delta1 = y1.*(1-y1).*e1;
```

The function `Softmax`, which the function `MultiClass` calls in, is implemented in the `Softmax.m` file shown in the following listing. This file implements the definition of the `softmax` function literally. It is simple enough and therefore further explanations have been omitted.

```
function y = Softmax(x)
  ex = exp(x);
  y  = ex / sum(ex);
end
```

The following listing shows the `TestMultiClass.m` file, which tests the function `MultiClass`. This program calls `MultiClass` and trains the neural network 10,000 times. Once the training process has been finished, the program enters the training data into the neural network and displays the output. We can verify the training results via the comparison of the output with the correct output.

```
clear all

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
```

The input data `X` of the code is a two-dimensional matrix, which encodes the white pixel into a zero and the black pixel into a unity. For example, the image of the number `1` is encoded in the matrix shown in Figure 4-14.

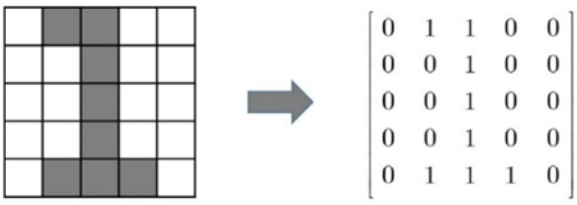

Figure 4-14. *The image of the number 1 is encoded in the matrix*

In contrast, the variable `D` contains the correct output. For example, the correct output to the first input data, i.e. the image of `1`, is located on the first row of the variable `D`, which is constructed using the one-hot encoding method for each of the five output nodes. Execute the `TestMultiClass.m` file, and you will see that the neural network has been properly trained in terms of the difference between the output and **D**.

So far, we have verified the neural network for only the training data. However, the practical data does not necessarily reflect the training data. This fact, as we previously discussed, is the fundamental problem of Machine Learning and needs to solve. Let's check our neural network with a simple experiment. Consider the slightly contaminated images shown in Figure 4-15 and watch how the neural network responds to them.

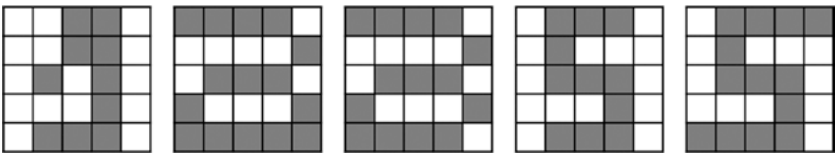

Figure 4-15. *Let's see how the neural network responds to these contaminated images*

The following listing shows the `RealMultiClass.m` file, which classifies the images shown in Figure 4-15. This program starts with the execution of the `TestMultiClass` command and trains the neural network. This process yields the weight matrices `W1` and `W2`.

```
clear all

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;
  y  = Softmax(v)
end
```

This code is identical to that of the `TestMultiClass.m` file, except that it has a different input `X` and does not include the training process. Execution of this program produces the output of the five contaminated images. Let's take a look one by one.

For the first image, the neural network decided it was a `4` by 96.66% probability. Compare the left and right images in Figure 4-16, which are the input and the digit that the neural network selected, respectively. The input image indeed contains important features of the number `4`. Although it appears to be a `1` as well, it is closer to a `4`. The classification seems reasonable.

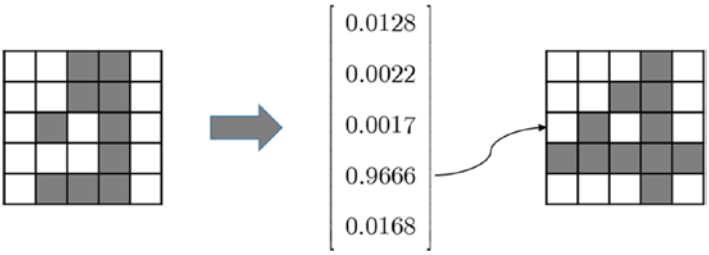

Figure 4-16. *Left and right images are the input and digit that the neural network selected, respectively*

Next, the second image is classified as a 2 by 99.36% probability. This appears to be reasonable when we compare the input image and the training data 2. They only have a one-pixel difference. See Figure 4-17.

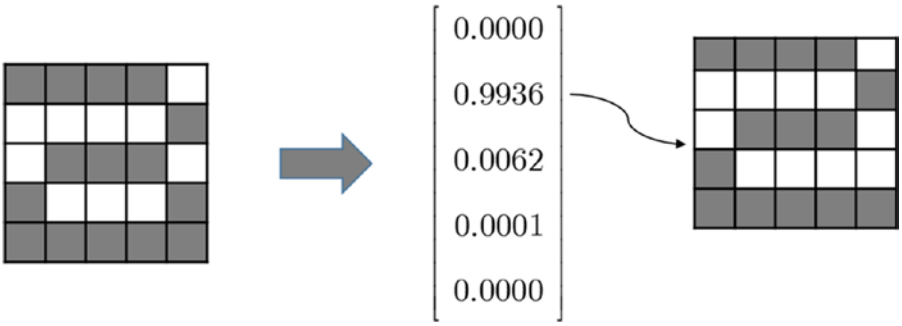

Figure 4-17. *The second image is classified as a 2*

The third image is classified as a 3 by 97.62% probability. This also seems reasonable when we compare the images. See Figure 4-18.

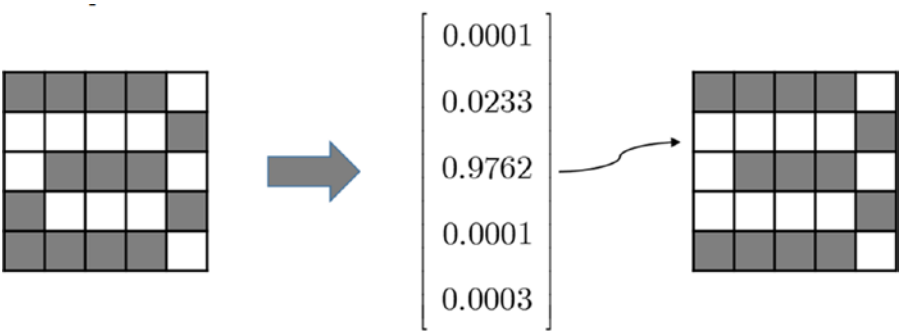

Figure 4-18. *The third image is classified as a 3*

However, when we compare the second and third input images, the difference is only one pixel. This tiny difference results in two totally different classifications. You may not have paid attention, but the training data of these two images has only a two-pixel difference. Isn't it amazing that the neural network catches this small difference and applies it to actual practice?

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.

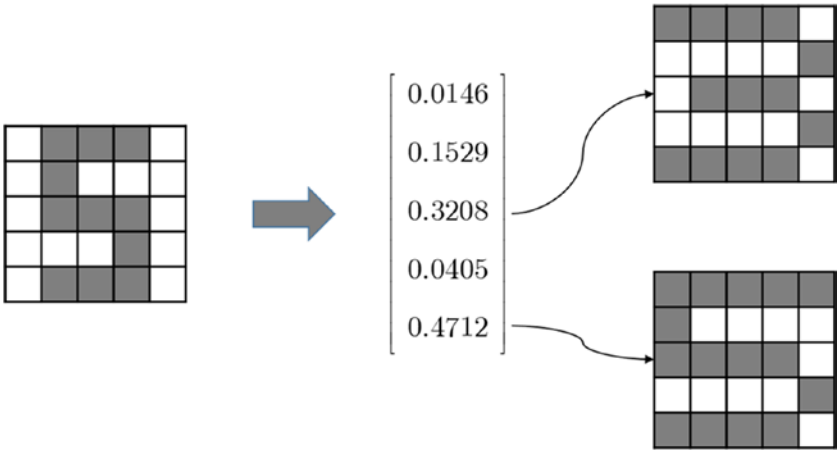

Figure 4-19. *The neural network may have to be trained to have more variety in the training data in order to improve its performance*

Finally, the fifth image is classified as a 5 by 98.18% probability. It is no wonder when we see the input image. However, this image is almost identical to the fourth image. It merely has two additional pixels on the top and bottom of the image. Just extending the horizontal lines results in a dramatic increase in the probability of being a 5. The horizontal feature of a 5 is not as significant in the fourth image. By enforcing this feature, the fifth image is correctly classified as a 5, as shown in Figure 4-20.

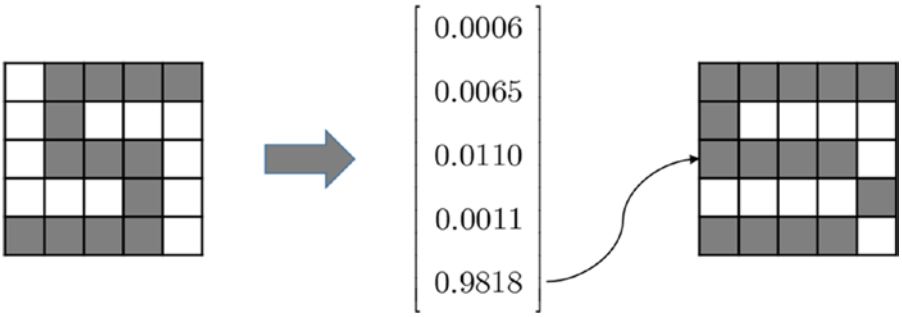

Figure 4-20. *The fifth image is correctly classified as a 5*

Summary

This chapter covered the following concepts:

- For the neural network classifier, the selection of the number of output nodes and activation function usually depends on whether it is for a binary classification (two classes) or for a multiclass classification (three or more classes).
- 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.
- For a multiclass classification, the neural network includes as many output nodes as the number of classes. The softmax function is employed for the activation function of the output node. The correct output of the training data is converted into a vector using the one-hot encoding method. The cost function of the learning rule employs the cross entropy function.