CHAPTER 4

Learning internal representations

Introduction

In the previous chapter, you trained a single-layered perceptron on the problems AND and OR using the delta rule. This architecture was incapable of learning the problem XOR (see Table 3.1 on page 31). To train a network on XOR you need a multi-layered perceptron like the one shown in Figure 4.1. These networks contain a layer of hidden

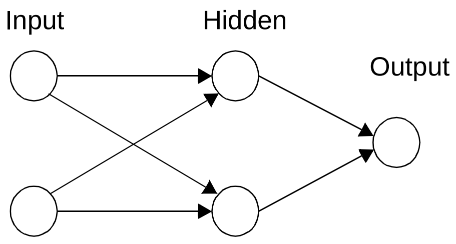


FIGURE 4.1 A simple multi-layered perceptron architecture for learning the Boolean function XOR

units that form "internal representations" of the input patterns. Connections propagate activity forward from the input nodes to the hidden nodes and from the hidden nodes to the output node. There are no connections between nodes within a layer or feedback connections to nodes in previous layers. Hence, network architectures of this type are often referred to as feedforward networks. During training, error is assigned to the hidden units by back propagating the error from the output. The error on the output and hidden units is used to change the weights on the second and first layer of connections respectively. Before continuing with the exercises in this chapter make sure that you are familiar with the exact procedure by which backpropagation adapts the weights in a network in response to an error signal. See "Learning" on page 10. You should also have completed Exercise 1.2 (c) on page 14.

Network configuration

Start up **`tlearn`** and Open a New Project called **`xor`**. **`tlearn`** opens three new windows called **`xor.cf`**, **`xor.data`** and **`xor.teach`**. Use the **`xor.cf`** file to create a network with the same architecture as that shown in Figure 4.1.

Exercise 4.1

- What are the contents of the three files?

Once you are confident that you have specified correctly all the necessary files (check your files with the answers to Exercise 4.1 on page 89) open the Training Options… dialogue box. Set the Learning rate parameter to 0.3 and the Momentum parameter to 0.9. Select a Random seed of 5 and set **`tlearn`** to Train randomly for 4000 sweeps.

Training the network

When you have specified all these training options open the Error Display and Train the network. If you have set all the training options as suggested, then you should observe an error display identical to that depicted in Figure 4.2. Check your training options if you do not observe this learning curve!

The error curve shows a sharp drop around the 2000 sweep mark. What has happened? First, let us examine the output activations for each of the input patterns after 4000 sweeps. The easy way to do this is to use the Verify network has learned option in the Network menu. The output from the network is shown in Figure 4.3. Again, if you do not observe these values in your own simulation, check that you have set the options in the Testing options… dialogue box correctly (see

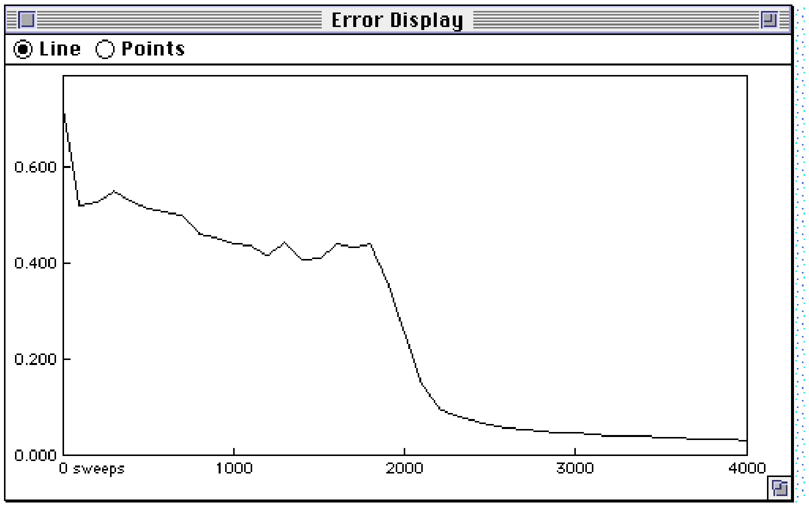
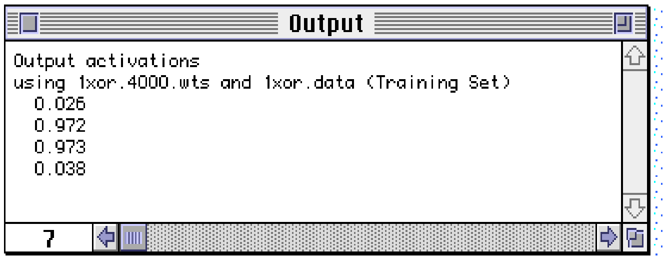


FIGURE 4.2 An error curve for a multi-layered perceptron trained on the XOR problem.



```
Output activations
using 1xor.4000.wts and 1xor.data (Training Set)
  0.026
  0.972
  0.973
  0.038
```

FIGURE 4.3 Output activations from multi-layered perceptron after training on Boolean XOR for 4000 sweeps.

Figure 3.20 on page 60 and accompanying text for instructions). Now consult the **`xor.teach`** file for the target activation values. The teacher signals for the four input patterns are **`0`**, **`1`**, **`1`**, **`0`** respectively. So the network has done very well at solving this problem. (Recall that we cannot expect the network to achieve exact matches to the target values—see Exercise 3.3 on page 63).

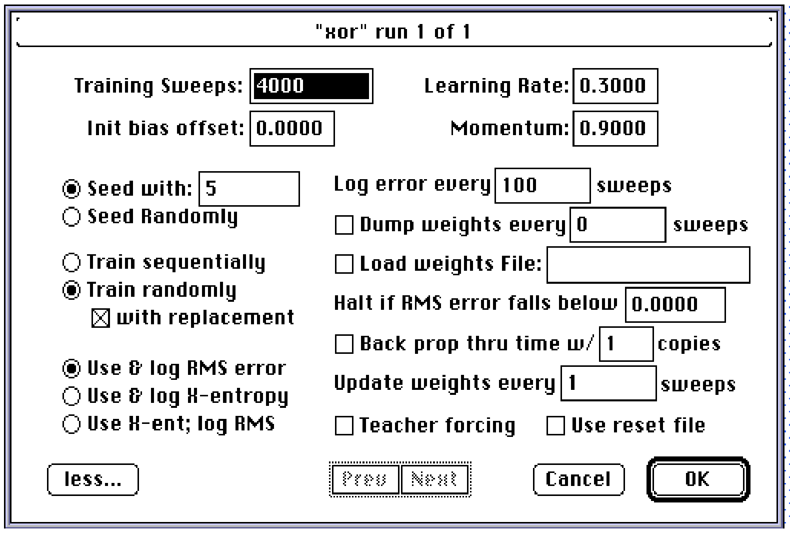
Exercise 4.2

- Can we tell from the error curve in Figure 4.2 if the network has solved XOR?

But why is there a sudden drop in error around the 2000 sweep mark? To answer this question, it would be useful to examine output activations prior to this point in training. We can achieve this by saving weight files during training of the network and then testing network output with these weight files instead of the network's final (mature) state.

Testing at different points in training

Open the Training options… dialogue box in the Network menu. Notice the more… button in the bottom left-hand corner of the dialogue box. Click on this button. The dialogue box expands to offer a wider range of training options as shown in Figure 4.4. We won't


"xor" run 1 of 1

Training Sweeps: 4000
Learning Rate: 0.3000
Init bias offset: 0.0000
Momentum: 0.9000

[✓ Seed with: 5]
Seed Randomly

Train sequentially
[✓ Train randomly]
[✓ with replacement]

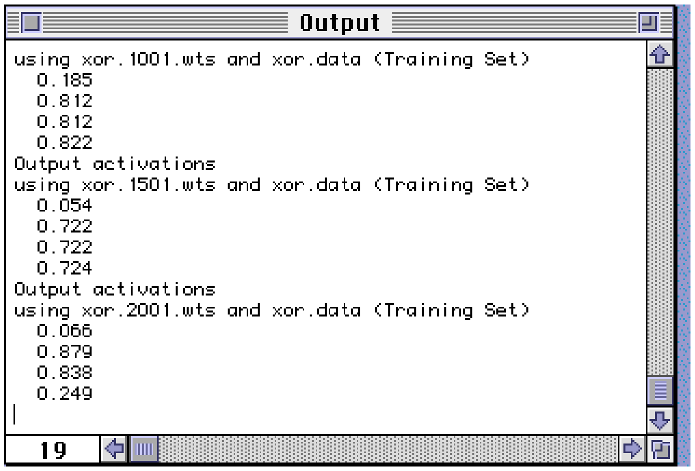
[✓ Use & log RMS error]
Use & log X-entropy
Use X-ent; log RMS

Log error every 100 sweeps
Dump weights every 0 sweeps
Load weights File:
Halt if RMS error falls below 0.0000
Back prop thru time w/ 1 copies
Update weights every 1 sweeps
Teacher forcing
Use reset file

FIGURE 4.4 The Training options… dialogue box expanded to reveal the full range of training options.

digress at this point to consider all the available options. For current purposes, the Dump weights option is of interest. Check this box and set the weights to be dumped every 500 sweeps. Leave all the other options as they are. Close the dialogue box and Train the network again. If you still have the Error display open, you should observe an identical error curve unfold. **`tlearn`** has performed an exact replication of the previous simulation using the same random seed (initial weight configuration) and same randomized order of pattern presentation. However, **`tlearn`** has saved weight files every 500 sweeps. Since the network has been trained for 4000 sweeps, 8 weight files will have been saved.

Next, open the Testing options… dialogue box and select the Earlier one: button. Click in the box to the right of the button to activate the dialogue box. Choose the **`xor.1501.wts`** file. Close the dialogue box and Verify the network has learned. **`tlearn`** will now display the output activations for the four input patterns after it has been trained for 1501 sweeps. By resetting the weights file in the Testing Options… dialogue box and Verifying the network has learned, you can determine output activations at any point in training. Figure 4.5



```
using xor.1001.wts and xor.data (Training Set)
  0.185
  0.812
  0.812
  0.822
Output activations
using xor.1501.wts and xor.data (Training Set)
  0.054
  0.722
  0.722
  0.724
Output activations
using xor.2001.wts and xor.data (Training Set)
  0.066
  0.879
  0.838
  0.249
```

FIGURE 4.5 Sample output activations after 1001, 1501 and 2001 sweeps of training on XOR.

shows the output activations after 1001, 1501 and 2001 sweeps respectively. Using the rounding off criterion, we observe that the network has learned the first three patterns correctly already after 1001 sweeps. Overall, however, the network is behaving as though it was supposed to learn Boolean OR, i.e., it categorizes the fourth input pattern in the same fashion as the second and third input patterns. By the 1501 sweep mark the output activities on the first and fourth patterns have been reduced—a move in the right direction. However, the reduction in error on these two patterns has been achieved (apparently) at the expense of the second and third patterns. Notice that the output activities for the second and third patterns have reduced instead of increasing. The network is still behaving like Boolean OR. Finally, after 2001 sweeps of training the output activity for the fourth pattern has decreased considerably and the activity for the second and third patterns has increased again. Activity patterns have begun to move in the right direction and using the rounding off criterion, the network has now solved XOR. Notice that the period of training between 1501 and 2001 sweeps corresponds to the sudden drop in global error depicted in Figure 4.2. What has happened in the network to bring about this change?

Examining the weight matrix

The response of a network to a given input is determined entirely by the pattern of connections in the network and activation functions of the nodes in the network. We know what the activation functions are—**tlearn** specifies them in advance. However, the network organizes its own pattern of connections—the weight matrix. **tlearn** possesses a useful facility for examining weight matrices—the Connection Weights option in the Display menu. Open this display. As we saw in Figure 3.17 on page 52, Connection Weights displays a Hinton diagram of the weight matrix. In this case, we want to examine the weights at different points in training. Open the Testing Options… dialogue box and make sure that the Earlier one: button is set to **xor.2001.wts**—double click on the box to activate the dialogue box permitting you to select the **xor.2001.wts** file. Close the Testing Options… dialogue box and select the Verify network has learned option. **tlearn** will display the output activations again and update Connection Weights to display a Hinton diagram for the state of the network after 2001 sweeps. You should obtain a Hinton diagram identical to that depicted in Figure 4.6.

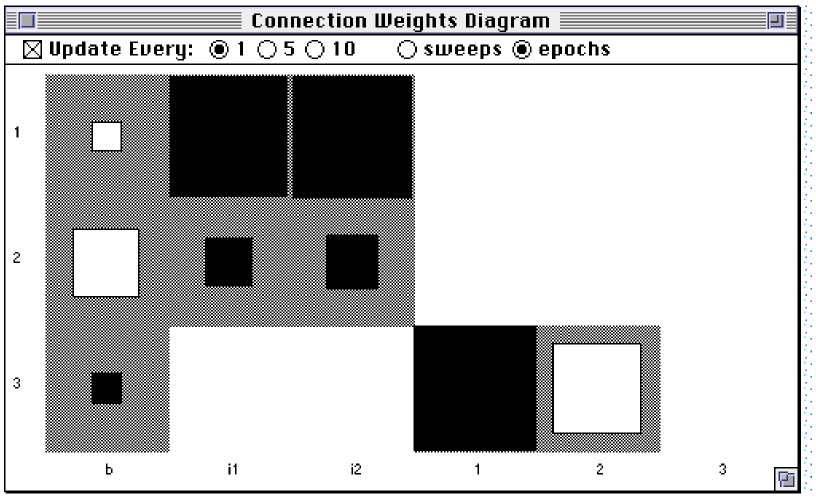

Connection Weights Diagram
Update Every: [✓ 1] 5 10 sweeps [✓ epochs]
1
2
3
b
i1
i2
1
2
3

FIGURE 4.6 Connection Weights displays a Hinton diagram of the weight matrix for XOR after 2001 sweeps. Rows represent receiving nodes and columns represent sending nodes. White squares represent positive weights. Black squares stand for negative weights. The size of the square reflects the absolute value of the connection.

Recall that the Hinton diagram organizes connections by rows and columns. The row identifies the receiving unit and the column identifies the sending unit. So row 1 in Figure 4.6 identifies all the connections going into node 1 (the first hidden node). There are three connections—from the bias and the two input nodes. There are no feedback connections from the first hidden node to itself or from the second hidden node or output node to the first hidden node. Therefore, these areas in the Hinton diagram are left blank. We observe in Figure 4.6 that the first hidden node has a mildly positive connection from the bias node, so it is disposed to become active in the absence of any other input. In contrast, the first hidden node receives strong inhibitory connections from both input nodes. Thus, any activation coming up the input lines will tend to turn the first hidden node off.

Exercise 4.3

- Draw a diagram of the network after 2001 sweeps depicting all connections (including the bias) as positive or negative.

In a later section we will examine how this network connectivity manages to provide a solution to the XOR problem. For the time being however we are concerned with the *changes* that occur in the network between 1501 and 2001 sweeps that enable the network to solve the problem. Now examine the weight matrix after 1501 sweeps of training. Open the Testing Options... dialogue box and set the Earlier one: option to **`xor.1501.wts`**. Close the dialogue box and Verify the Network has learned. The Connection Weights Diagram will be updated to depict a Hinton diagram for the weight matrix at 1501 sweeps as shown in Figure 4.7.

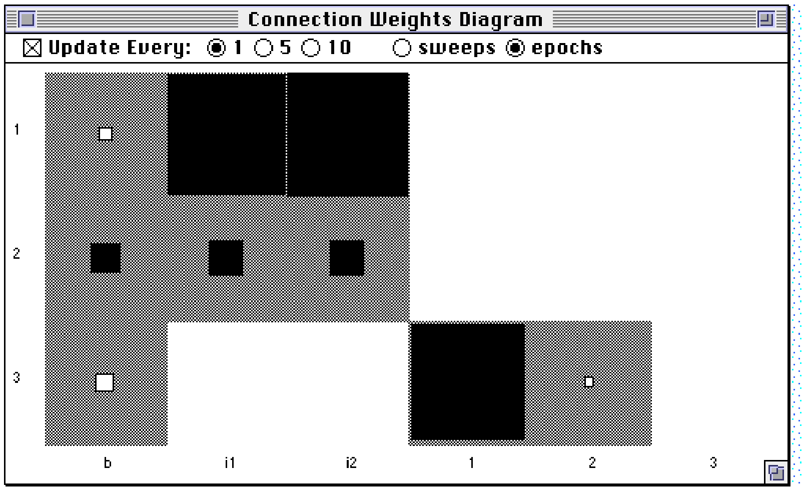

Connection Weights Diagram
Update Every: [✓ 1] 5 10 sweeps [✓ epochs]
1
2
3
b
i1
i2
1
2
3

FIGURE 4.7 Connection Weights displays a Hinton diagram of the weight matrix for XOR after 1501 sweeps.

How does the weight matrix shown in Figure 4.7 change into the weight matrix shown in Figure 4.6? The most significant changes take place in the connections from the bias node to the hidden and output nodes, and from the second hidden node (node 2) to the output node

(node 3). The remaining connections do not change much. In particular, the bias connections to the hidden nodes grow in strength to provide positive excitation to their target nodes while the bias connection to the output node switches to inhibit its activation. The connection from the second hidden node to the output node increases its capacity to excite the output node. These changes take place in unison, as they must if they are to permit the fourth input pattern to be classified correctly.

Exercise 4.4

- Draw a diagram of the network depicting the exact values of the connections (to one decimal place) at 1501 and 2001 sweeps. Can you see how the network fails to solve the XOR at 1501 sweeps but passes the test at 2001 sweeps? *Hint: You will need to examine the weight files themselves as shown in Figure 3.26 on page 73.*

Hinton diagrams provide a convenient overview of the connectivity in the network. You can even request that **`tlearn`** displays changes in the weight matrix on-line. Try it. Make sure that the Connection Weights diagram is displayed and then simply Train the Network. Initially, you will observe substantial swings in the weight values in the network. However, weight values will gradually stabilize. If you have the Error Display active then you can also observe how changes in the global RMS error coincide with changes in the weight matrix. The next stage in analyzing network performance involves examining hidden node activations.

Hidden node representations

The activations of the hidden nodes provide an important clue as to how the network has solved XOR. In general, we may consider the hidden unit activations associated with an input pattern to be the network's *internal representation* of that pattern. We shall discover that patterns that look quite dissimilar at the input layer can be almost identical when we view their activations at the hidden unit level. Conversely, patterns that are similar at the input layer can end up looking quite different at the hidden layer. Hidden units and the connections feeding into them have the capacity to transform the similarity relations of the patterns in the input space so that the nature of the problem itself is changed.

We shall investigate several ways of examining these activities in the network. To begin with open the Testing Options… dialogue box and set the Earlier one: option to **xor.2001.wts**. Close the dialogue box and open the Node Activation display in the Display menu. Click on the **First Data Pattern** button. The display should update to reveal the display in Figure 4.8. This represents the pattern of

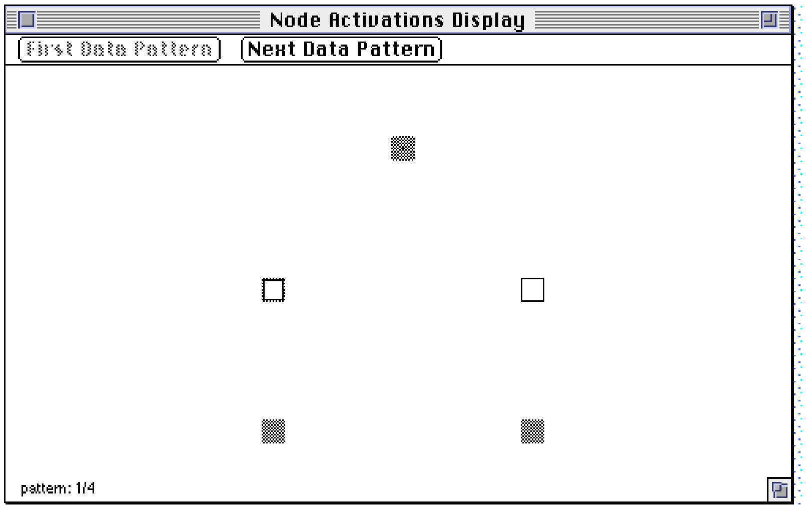


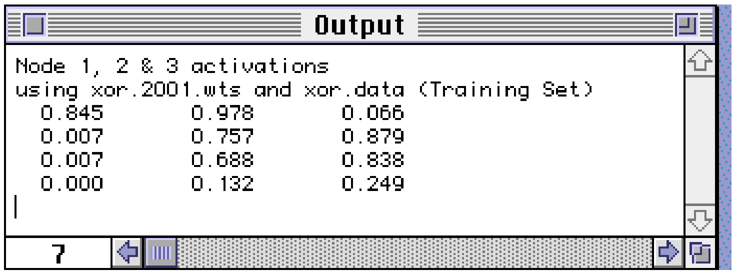
FIGURE 4.8 Node Activation Display depicts the activation of all nodes in the network (excluding the bias). The level of activation of each node determines the size of the white square. A grey node indicates an inactive node. In the current display, the input and output nodes are inactive and the hidden nodes are active.

activity in the network when the first input pattern 0 0 is presented to the network. The current pattern is identified in the bottom left-hand corner of the display. All the nodes in the network are represented (except for the bias node). The level of activation of a node is represented by the size of the white square—large squares reflect high activity. Grey squares indicate dormant nodes. Input nodes are displayed at the bottom of the diagram. Subsequent layers in the network are depicted at higher levels in the display. Hence, in Figure 4.8, the input nodes are dormant, the output node is dormant (as it should be for correct performance) and the hidden nodes are active. By clicking on the Next Data Pattern button, you can display node responses to all of the four input patterns.

Exercise 4.5

- Examine the activation of the hidden nodes in response to the different input patterns. Can you see how the network is solving the non-linear XOR problem? Why is it that the first input pattern produces highly active hidden nodes but a dormant output node?

Just as it is possible to determine the exact values of the connection weights in the network, it is also possible to determine the exact activations of the hidden nodes. Open the **xor.cf** file using the Open option from the File menu (if it isn't already open). Observe that in the SPECIAL: section there is a line with the instructions: selected = 1–3. This line tells **tlearn** to output the activations of the selected nodes whenever the Probe Selected Nodes option from the Network menu is chosen. Make sure that the Testing Options... is still set to use the weights file **xor.2001.wts** and then select Probe Selected Nodes. **tlearn** will display activations for nodes 1–3 for the four input patterns as shown in Figure 4.9. Notice that the third



```
Node 1, 2 & 3 activations
using xor.2001.wts and xor.data (Training Set)
  0.845       0.978       0.066
  0.007       0.757       0.879
  0.007       0.688       0.838
  0.000       0.132       0.249
```

FIGURE 4.9 Node activations in the Output window for nodes 1–3 in a network trained on XOR for 2001 sweeps. The output is produced by selecting the Probe Selected Nodes option in the Network menu. Nodes are selected in the **SPECIAL:** section of the **xor.cf** file.

column of activation values is identical with the activations displayed in Figure 4.5 for 2001 sweeps. The first two columns in Figure 4.9 list the activations of the two hidden nodes for the four input patterns. These values give the precise values used to calculate the Node Activations display in Figure 4.8.

Exercise 4.6

1. Use the weight matrix from **`xor.2001.wts`** (see Exercise 4.4) to confirm that the activation values listed in Figure 4.9 are correct. *Hint: You will need to refer to Exercise 1.1 on page 8 to convert the net input to a node into its activation value.*
2. Have we explained the *sudden* decrease in the global error score observed in Figure 4.2?

Role of learning rate and momentum

The learning rate parameter η (see Equation 1.4 on page 12) determines what proportion of the error derivative is used to make changes to the weights in the network. In the Training Options... dialogue box you are also given the option of specifying the value of another parameter called momentum (μ). The momentum parameter provides another means for manipulating the weights but based on the changes that were made on the previous learning trial. The value of the momentum parameter determines what proportion of the weight changes from the previous learning trial will be used on the current learning trial. Mathematically, we can express this notion as

$$\Delta w_t = -\eta\frac{\partial E}{\partial w} + \mu\Delta w_{t-1} \qquad \textbf{(EQ 4.1)}$$

which reads: "The change in the weight at time t is equal to the learning rate η multiplied by the negative slope of the error gradient with respect to the weight, plus the momentum μ multiplied by the size of the weight change on the previous learning trial." Thus, if the weight change on the previous learning trial is large and μ is large, the weight change on the current learning trial will also be large even if the derivative of the error with respect to the weight (i.e., $\partial E/\partial w$) is small. In other words, the momentum term has the effect of maintaining weight change in the network when the error surface flattens out and speeding up weight changes when the error surface is steep. Of course, if $\mu = 0$ then the learning algorithm operates in the usual fashion.

Exercise 4.7

1. It has been suggested that momentum can help the learning algorithm solve problems like XOR which would otherwise be very difficult to solve. Do you agree with this suggestion? Under what conditions might it be true?
2. Try running the XOR simulations again but experiment with different values of learning rate and momentum. Can you find an optimum combination of learning rate and momentum for the XOR problem? In general, is it best to have high or low levels of learning rate and momentum for XOR? When you have found another combination of learning rate and momentum that solves XOR, analyze the solution the network has found and compare it with the solution achieved in this chapter. Is there only one solution to XOR?

Role of hidden nodes

You have already discovered the importance of hidden nodes for solving the XOR problem. Now examine the different ways in which the quantity and organization of hidden nodes can effect network performance.

Batch versus pattern update

Examine the expanded version of the Training Options... dialogue box. You will notice that there is an option called Update weights every: which has been set to 1 sweep. This option tells **`tlearn`** when to make weight changes in the network. Until now, we have used the default setting for this option which is to update the weights after

Exercise 4.8

1. Reconfigure your network to use different numbers of nodes in the hidden layer, say from 1 to 5. Does increasing the number of hidden units assist the network in solving the task? Determine the role of the individual units in solving the task.
2. Configure the network with an additional layer of 2 hidden nodes. Does the additional layer accelerate learning on XOR? Examine the internal representations constructed in the different hidden layers.

every pattern presentation. In network jargon, this is called pattern update. Of course, it is possible to delay making weight changes until the network has seen the other training patterns. A common training regime, called batch update, involves presenting all input patterns just once, accumulating the calculated changes for each connection and then updating the weights for all pattern presentations simultaneously. In the XOR problem, this is equivalent to setting Update weights every: to 4 sweeps.

Exercise 4.9

- What do you think are the consequences for the learning profile of the network when choosing between pattern and batch update? Can you think of any computational advantages in using batch mode? Test your theory by running XOR in batch mode. *Note: You will also need to deactivate the* With Replacement *under the* Train randomly *check box if you wanted to guarantee that the network sees each training pattern on every epoch. Of course, direct comparison with a pattern update training schedule would then be inconclusive since until now you have selected patterns randomly with replacement.*

Answers to exercises

Exercise 4.1

- The **`xor.cf`** file should contain the following information:

```
NODES:
nodes = 3
inputs = 2
outputs = 1
output node is 3
CONNECTIONS:
groups = 0
1-2 from i1-i2
3 from 1-2
1-3 from 0
SPECIAL:
selected = 1-3
weight_limit = 1.00
```

The **`NODES:`** section indicates that there are 3 nodes in the network—2 hidden nodes and 1 output node (input nodes don't count). The **`CONNECTIONS:`** section contains an additional line for the extra layer of connections in the multi-layered perceptron. There is also a connection from the bias to the hidden nodes and the output node (**`1-3 from 0`**). Notice how it is possible to specify network architectures of increased complexity through minor additions to the **`.cf`** file.

The **`xor.teach`** file requires only a minor change in relation to the **`or.teach`** file:

```
distributed
4
0
1
1
0
```

The **`xor.data`** file is identical to the **`or.data`** file.

Exercise 4.2

- Whether the network has solved the problem depends on the error criterion you decide to use. If you adopt the “rounding off” procedure introduced in Exercise 3.3 on page 63 and further discussed in Exercise 3.4 on page 65 then a level of global RMS error, namely 0.25, can be used to *guarantee* successful performance on all four input patterns. The global RMS error displayed in Figure 4.2 falls below this value around the 2000 sweep mark, so the error curve alone tells us that the network has solved the problem. Of course, the network may have solved the problem before this point in training.

Exercise 4.3

- Figure 4.12 shows the connectivity of the network after 2001 sweeps of training on the XOR probelm. Each connection is depicted as inhibitory (-) or excitatory (+).

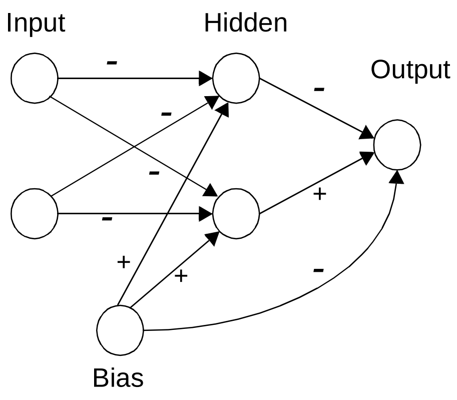
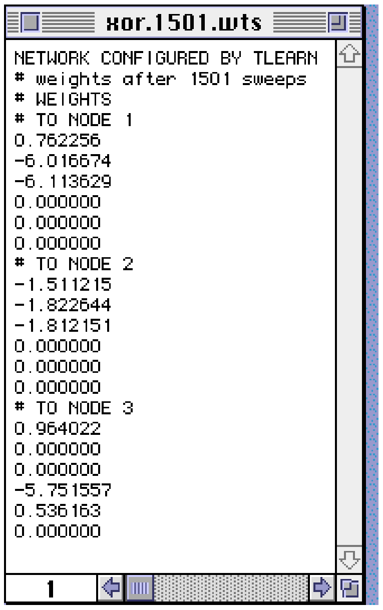


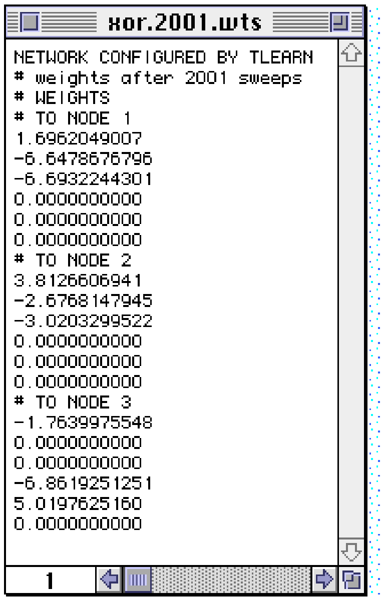
FIGURE 4.10 Connections weights in multi-layered perceptron trained for 2001 sweeps on XOR.

Exercise 4.4

- Use the weight files **`xor.1501.wts`** and **`xor.2001.wts`** to obtain the exact values of the connections in the network. You can Open these files from the File menu. Their contents are shown in Figure 4.11. Con-



```
xor.1501.wts
NETWORK CONFIGURED BY TLEARN
# weights after 1501 sweeps
# WEIGHTS
# TO NODE 1
0.762256
-6.016674
-6.113629
0.000000
0.000000
0.000000
# TO NODE 2
-1.511215
-1.822644
-1.812151
0.000000
0.000000
0.000000
# TO NODE 3
0.964022
0.000000
0.000000
-5.751557
0.536163
0.000000
```



```
xor.2001.wts
NETWORK CONFIGURED BY TLEARN
# weights after 2001 sweeps
# WEIGHTS
# TO NODE 1
1.6962049007
-6.6478676796
-6.6932244301
0.0000000000
0.0000000000
0.0000000000
# TO NODE 2
3.8126606941
-2.6768147945
-3.0203299522
0.0000000000
0.0000000000
0.0000000000
# TO NODE 3
-1.7639975548
0.0000000000
0.0000000000
-6.8619251251
5.0197625160
0.0000000000
```

FIGURE 4.11 Weight files after 1501 and 2001 sweeps of training on XOR.

nections are listed for each target node. The connection from the bias is listed first, then connections from the input units and so on up through the network. A value of 0.000000 will almost always indicate that the connection is non-existent. The state of the network after 2001 sweeps of training is shown in Figure 4.12.

Exercise 4.5

- Both hidden units are highly active for the first input pattern (0 0) while only the second hidden unit is active for the second and third input patterns (1 0) and (0 1). Neither of the hidden units are active for the last

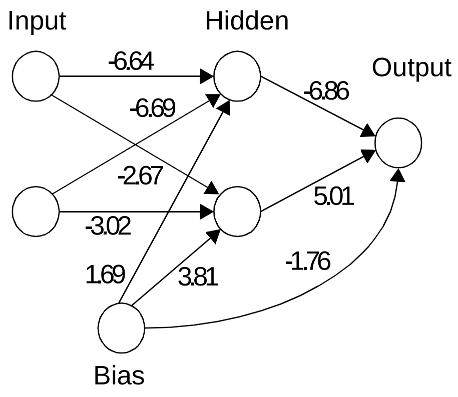


FIGURE 4.12 Connections weights in multi-layered perceptron trained for 2001 sweeps on XOR.

input pattern (1 1). In other words, the internal representations of the second and third input patterns are more or less identical and so will produce the same output activation (1)—as they should. In this case, the excitatory activity from the second hidden node is sufficient to overcome the negative effect of the bias and turn on the output unit. For the first and fourth input patterns, the negative connections from the bias and the first hidden unit guarantee that the output unit remains switched off. The layer of hidden units have transformed a *linearly inseparable* problem into a *linearly separable* one (see Chapter 2 of the companion volume, *Rethinking Innateness*, for a discussion of *linear separability*).

Exercise 4.6

1. Repeat the procedure you performed in Exercise 3.6 on page 66. Note, however, that in this case you will also need to use the activations of the hidden units to calculate the activations of the output unit for the 4 input patterns. Don’t forget to include the bias in your calculations!

2. Although we have extracted a good deal of information from `tlearn` as to how the network solves XOR, we have not explained why there is a sudden decrease in the global error between 1501 and 2001 sweeps. There are several ways to understand how the network suddenly finds a solution to the XOR problem.

The learning algorithm that we use in this book, backpropagation, belongs to a family of *gradient descent* learning algorithms. The idea is that the learning algorithm attempts to decrease the error at the output by traversing an *error landscape* in a downhill direction. It achieves this by calculating the slope of the error surface at the current location and making changes in the weights which will take it downhill. This is the calculation embodied in in the partial derivative introduced in Equation 1.4 on page 12. The idea of gradient descent is discussed in more detail in the companion volume *Rethinking Innateness*, Chapter 2 (page 71). The reason why the network suddenly finds a solution to XOR is that the error surface has a sharp dip in it, such that a small change in the value of the weights brings about a relatively large change in the output error.

Another way to understand how the network suddenly finds a solution to XOR requires that you understand the idea of *linear separability* (also discussed in more detail in *Rethinking Innateness*, Chapter 2 (page 62–63). We'll review the problem briefly again here. It is possible to visualize the Boolean functions AND, OR and XOR graphically in a two dimensional plane as shown in Figure 4.13. Different partitionings of the

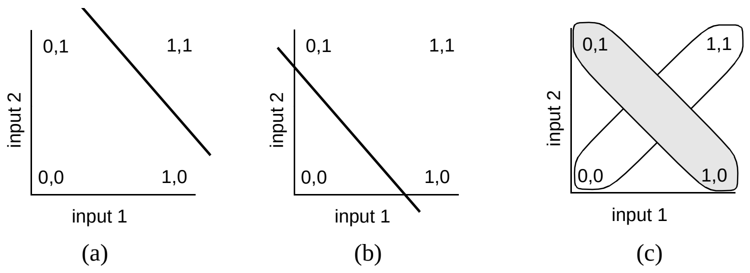


FIGURE 4.13 Geometric representation of the XOR problem. If the four input patterns are represented as vectors in a two-dimensional space, the problem is to find a set of weights which implements a linear decision boundary for the output unit. In (a), the boundary implements logical AND. In (b), it implements logical OR. There is no linear function which will simultaneously place 00 and 11 on one side of the boundary, and 01 and 10 on the other, as required for (c).

space correspond to different Boolean functions. The partitionings shown in Figure 4.13 (a) and (b) represent Boolean AND and OR, respectively. Notice how the space can be partitioned appropriately simply by drawing a line separating three patterns from the other one. The problems are linearly separable. In fact, there are an infinite number of lines that you could draw that would partition the space appropriately for these two problems (different angles and positions). Each of these lines corresponds to a different solution that the network might find, i.e., a different configuration of the weight matrix. In contrast, XOR is linearly non-separable—it is not possible to partition the space with a single line such that all patterns are placed in the correct partition. This is why you

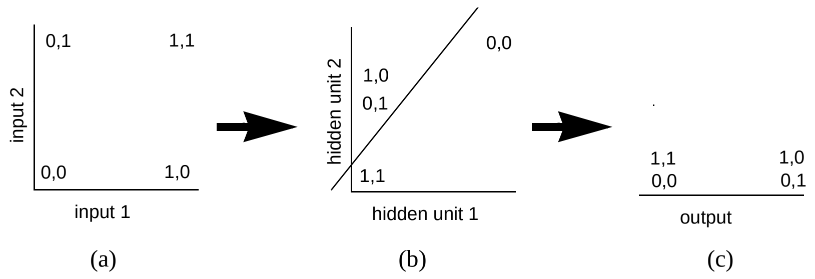
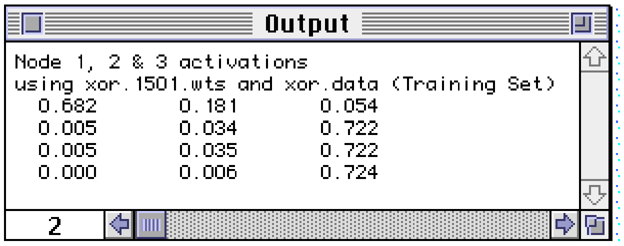


FIGURE 4.14 Transformation in representation of the four input patterns for XOR. In (a) the similarity structure (spatial position) of the inputs is determined by the form of the inputs. In (b) the hidden units “fold” this representational space such that two of the originally dissimilar inputs (0,1 and 1,0) are now close in the hidden unit space; this makes it possible to linearly separate the two categories. In (c) we see the output unit’s final categorization (because there is a single output unit, the inputs are represented in a one-dimensional space). Arrows from (a) to (b) and from (b) to (c) represent the transformation effected by input-to-hidden weights, and hidden-to-output weights, respectively.

need a network with hidden units to solve the problem. As we saw in Exercise 4.5 on page 91, the hidden units transform a linearly non-separable problem into a linearly separable one. However, the hidden units cannot do this immediately. The connections between the input units and the hidden units (and the bias) have to be adapted to an appropriate configuration by the learning algorithm. The configuration is shown in Figure 4.12.

Think about this again from a two dimensional point of view. We saw in Exercise 4.5 that the network treats the input patterns (1 0) and (0 1) as more or less identical at the hidden unit level. This is represented graphically in Figure 4.14. Once the hidden unit activations corresponding to the four input patterns become linearly separable in accordance with the problem at hand, the network can find a solution. This corresponds to the hidden unit activation of one of the input patterns (in this case 1 0) moving into a region of the space occupied by its partner (in this case 0 1). A line can then partition the hidden unit activation patterns into their appropriate categories and produce the correct output activations. Before this point, no solution is possible and the error remains high.

For example, after 1501 sweeps of training in the network, the activations of the hidden and output nodes can be probed to yield the activation values shown in the Output window shown in Figure 4.15. The



```
Node 1, 2 & 3 activations
using xor.1501.wts and xor.data (Training Set)
  0.682       0.181       0.054
  0.005       0.034       0.722
  0.005       0.035       0.722
  0.000       0.006       0.724
```

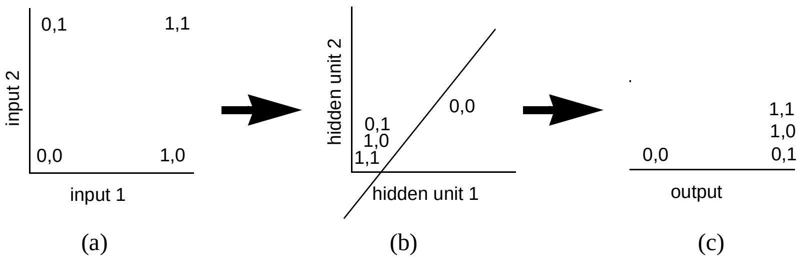


FIGURE 4.15 Transformation in representation of the four input patterns for XOR after 1501 sweeps of training. The network is responding as if it had been trained on Boolean OR. The network achieves the solution depicted in Figure 4.14 by increasing the activation of the second hidden unit for input patterns (1 0) and (0 1) and shifting the demarcation of the linear partition.

corresponding positions of the patterns in two dimensional space are also shown in Figure 4.15. Clearly, the hidden unit activations do not permit a linear separation of the problems. In fact, the network is behaving as though it had been trained to solve Boolean OR. To solve XOR the second hidden unit activations for the patterns (1 0) and (0 1) must be higher. Once the network has achieved this, the problem becomes linearly separable and the solution is quickly reached.

Exercise 4.7

1. Generally, momentum helps when the patterns in the training set have to be treated in similar ways. Learning on previous patterns can be transferred to subsequent patterns. Sometimes, momentum can help the network avoid getting trapped in local minima—local regions on the error landscape where the error gradient is zero (see *Rethinking Innateness*, Chapter 2, page 71).

2. The identification of an optimum combination of learning rate and momentum depends on the start state of the network. Since this is often determined in a random fashion, it is difficult to specify what the appropriate values of these parameters should be for any given simulation. However, it is generally a good idea to use a small learning rate to avoid situations where new learning wipes out old learning.

   You will find that there are many solutions to XOR. Some of them are quite surprising!

Exercise 4.8

1. You will find that the network seems to do best with just 2 or 3 hidden units. Although there are many solutions available to the network with 5 hidden units, a common solution is to turn off the hidden units that the network doesn't need by developing strong inhibitory connections to them from the bias node. In this manner, the network comes to behave as if it only had 2 or 3 hidden units.

2. Adding extra layers of hidden units to the network rarely speeds up the time to solution. On the contrary, the extra layers of randomized weights obstructs the construction of good internal representations since the backpropagation of error (assignment of blame) is reduced from one layer to the next (see Equation 1.6 on page 14).

Exercise 4.9

- Since weight changes made in batch update reflect the *average* error for all the patterns in the training set, learning is likely to follow a smoother curve than pattern update. Training will also be faster in batch mode because fewer weight changes are being made on each epoch of training.