

## *Introduction*

---

Feedforward networks are trained to approximate a function associating a set of input patterns with a set of output patterns. It is also possible to teach networks simply to reproduce their input. In this case, the input and output layers have the same number of elements. The network is given a pattern and its task is to pass that pattern through itself until it is reproduced on the output layer. Thus, the input and the teacher pattern are identical. In such circumstances, the network is said to be doing *autoassociation* or *identity mapping*. In this chapter, you will:

- Learn how autoassociation can be used to convert a “local” representation into a “distributed” representation. This is sometimes called the *encoding* problem.
- Investigate how autoassociation can also be useful in discovering redundancies in a set of input patterns which lead to more efficient featural representations.
- Show how autoassociators can perform efficient pattern completion in the face of noisy input.
- Develop additional techniques for analyzing hidden node representations.

### *Local and distributed representations*

---

In this section, you will encode 4 bit vectors. Therefore, you will need a network containing 4 input nodes and 4 output nodes. Create a **New Project...** called **auto1**. Configure the **auto1.cf** with 2 hidden nodes, such that the 4 input nodes are all connected to the 2 hidden nodes and the 2 hidden nodes are all connected to the 4 output nodes in a strictly feedforward fashion. Connect a bias to the hidden and output nodes. The weights should be randomized within the range  $\pm 0.5$ .

Each pattern consists of a 4-bit vector (each bit will be either a 1 or a 0). Furthermore, the patterns you will use in this example are what might be called “local representations.” These are the patterns:

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

#### *Exercise 5.1*

---

- What is meant by “local representation?” In what sense are the above patterns instances of local representations?
- 

Create an **auto1.data** file and **auto1.teach** file appropriate for the task. Note that **tlearn** offers an alternative format for data presentation when the patterns are represented in a localist fashion, i.e., when only a few of many input nodes are non-zero. In the “localist” format, the input patterns in the **.data** file are a list of nodes, specifying only the numbers of those nodes whose values are to be set to one. The node specification follows the conventions used in the **.cf** file. See “The Configuration (**.cf**) file” on page 38. All other input nodes are assumed to be zero. Localist coding of the output patterns in the **.teach** file can be used to specify the output nodes which are to be on. In other words, an output pattern in the localist case is a set of integers designating the nodes whose target values are one. All other nodes are assumed to be zero. Possible **data** and **teach** files are shown in Figure 5.1. You may continue to use the “distributed” mode

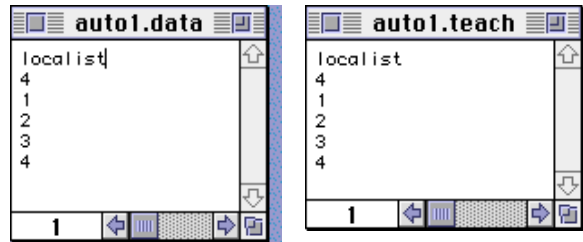


FIGURE 5.1 The `auto1.data` and `auto1.teach` files using localist coding mode.

of data coding if you wish. It is introduced here purely as a notational variant.

Check that you have set up your configuration file correctly by displaying the Network Architecture. `tlearn` should depict a network with the same pattern of connectivity as displayed in Figure 5.2. If you do not obtain this display, then check your `auto1.cf` file.

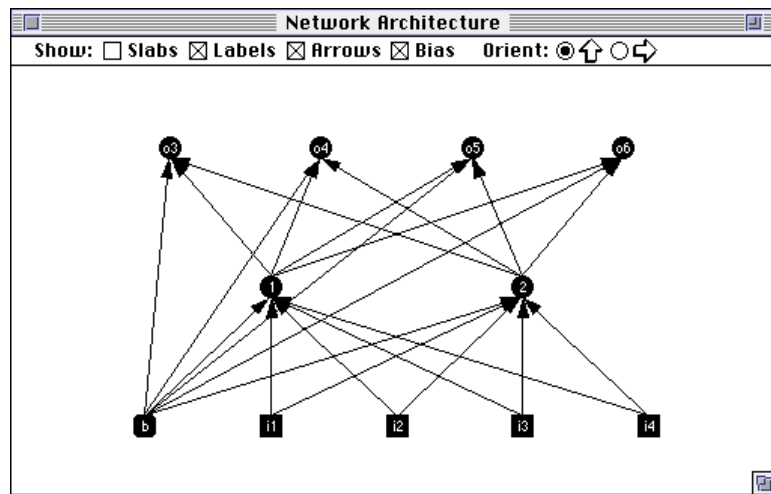


FIGURE 5.2 Network Architecture for `auto1`.

### Training the network

Feel free to experiment with the range of training options available to you. For purposes of exposition, however, we will refer to a simulation in which the following training options have been selected: Set the learning rate parameter to 0.3 and specify a momentum of 0.9. Specify a random seed of 1 and train randomly, updating the weights after every sweep (pattern update). Train the network for 3000 sweeps. You may also wish to activate the error display to examine the global RMS error.

#### *Exercise 5.2*

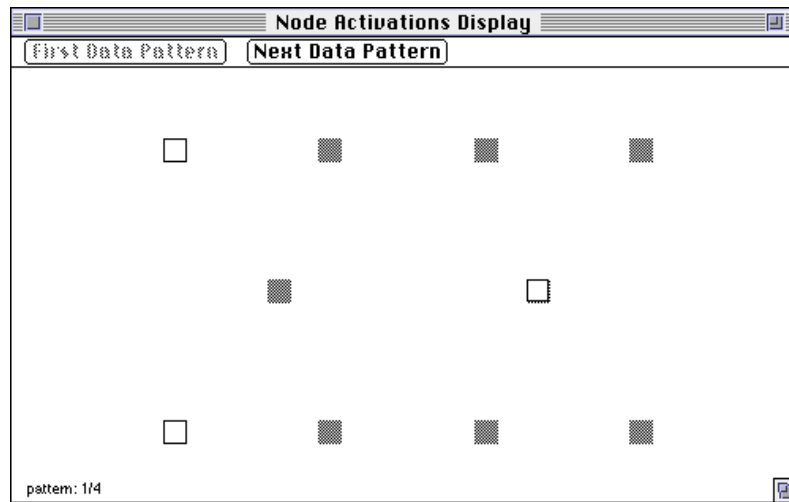
- 
- What level of global error guarantees that **tlearn** has found a solution to the encoding problem? (Assume the rounding off criterion for evaluation purposes.)
- 

When you have found a configuration of training parameters that yields a solution to the encoding problem, select the **Node Activation** display. Make sure that you have selected the **Most recent** weights file in the **Testing Options...** dialogue box. You should observe a display like that shown in Figure 5.3. Cycle through the four input patterns and check that the network is autoassociating in the required fashion. Cycle through the four input patterns again and see how the hidden units have encoded these 4 patterns. The hidden unit activations will have learned to encode their input/output pattern pairs by using a distributed representation of the patterns.

#### *Exercise 5.3*

- 
1. What is the key to this distributed representation? Why is the representation “distributed”?
  2. Open up the weights file **auto1.3000.wts** and look at the network which did the encoding. Draw the network. Show weights on each of the connections, and put the biases inside the nodes.
- 

You have seen how a network can represent 4 input patterns using 2 bits. What do you think would happen if you modified the network so



**FIGURE 5.3** Node Activations Display for **auto1** after 3000 training sweeps. that it had 5 inputs and 5 outputs, but still only 2 hidden units; and then trained it to encode the following patterns?

```

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

```

#### **Exercise 5.4**

- 
- Do you think the network could do the task? Construct such a network and report your results. So as not to override the work on **auto1**, call this project **auto2**.
-

### *Pattern completion*

---

Now return to the original autoassociation task in this exercise—**auto1**. Modify your **auto1.data** file so that one of the input patterns is distorted. For example, you might change the input pattern **1 0 0 0** to **0.6 0 0 0.2**. You’ve just introduced noise into the input stream. In the Testing Options... dialogue box select the **auto1.3000.wts** file which you created earlier. Now test your network again using the Verify Network has learned option. Alternatively, you can observe the response on the Node Activation Display.

#### *Exercise 5.5*

---

- Has the network partially reconstructed the noisy pattern? Experiment with input patterns that have noise in different locations and in differing amounts. Is there some critical noise level (or combination of noise values) for which the pattern completion fails?
- 

This phenomenon, called *pattern completion*, is a result of the compressed internal representation. The difference between a degraded and a pristine input pattern is partially eliminated when the activation goes through the nonlinearity at the hidden layer.

### *Feature discovery*

---

In this section, you will investigate another purpose to autoassociation, which is to discover a featural representation of the input. Because the intermediate (hidden) layer(s) are generally of smaller dimensionality than the input and output layers, for the autoassociation to work, the network is required to funnel the pattern of input activation through its narrow “waist.” The pattern of hidden unit activations must be efficient and concise; if the hidden units are to successfully regenerate the input pattern on the output layer, they must discover some representation of the input which is reduced but which contains all the information necessary to recreate that input. This task

encourages the network to discover any structure present in the input so that the input can be represented more abstractly and compactly. The network need know nothing about the significance of the input.

### Exercise 5.6

- 
- What does it mean to say that the concept of “redundancy” is a property of a *set of patterns*?
- 

To do this exercise, you will need a network that has 7 input nodes and 7 output nodes, and 3 hidden nodes. The **data** file should contain the following 8 input patterns:

```
1 1 0 1 0 0 1
1 0 0 1 1 1 1
1 0 1 1 1 0 0
1 1 0 1 0 0 0
0 1 0 0 0 0 1
0 1 0 0 0 1 0
0 0 0 0 1 0 0
0 0 1 0 1 1 0
```

Since the task is autoassociation, the input and output patterns will be identical. You will need to study the activation of the hidden nodes so make sure that they are **selected** in the **SPECIAL:** section of the **.cf** file.

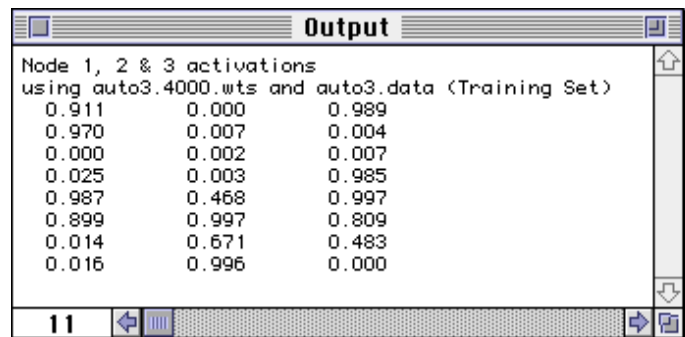
### Exercise 5.7

- 
1. Before running the network, study the input patterns. Do you see any way(s) of grouping the patterns? That is, do any of the patterns resemble each other? Are there different ways of grouping the patterns?
-

**Exercise 5.7**

- 
2. Train the network for 4000 sweeps. (The results reported below use a random seed of 1, learning rate of 0.3 and momentum of 0.9. However, you may choose a different configuration if you wish. Just remember that in this case the results will look a bit different.)
  3. After the network is trained, test the network to see that it has learned to generate the correct output. Has the network learned to regenerate the input patterns? How well?
- 

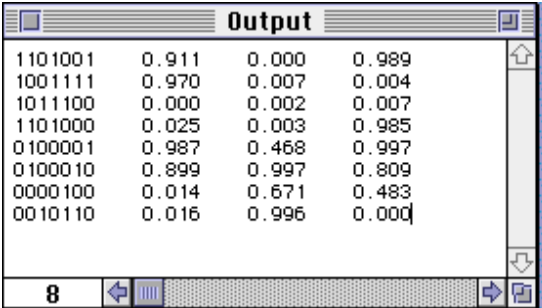
Now see if there is any similarity structure to these patterns. This involves looking at the hidden node activation patterns. Do this twice. First, test the activations of the 3 hidden nodes in response to each of the 8 probes. You can display hidden node activations by selecting the **Probe Selected Nodes** option in the **Network** menu. If you have specified the hidden nodes as “selected” in the **SPECIAL:** section of the **.cf** file, then the hidden node activations will be displayed in the **Output** window as shown in Figure 5.4. Can you discern any patterns? Probably not; so do something else. This involves editing the **Output** window containing the activation of the hidden nodes. First, clear the two lines at the beginning of the file containing header information so that you are left with a file containing 8 rows and 3 columns, each row



**FIGURE 5.4** Hidden node activations in an autoassociation task produced by **Probe Selected Nodes** in the **Network** menu.



specifying the hidden node activations for each input pattern. Insert a name representing the input pattern at the beginning of each line. You can use any numeric characters you choose for this label just so long as it doesn't contain a space. When you have edited the **Output** window it should resemble the window displayed in Figure 5.5. Use the **Save as...** option in the **File** menu to save this file as **auto3.sort**.



Input Pattern	Hidden Node 1	Hidden Node 2	Hidden Node 3
1101001	0.911	0.000	0.989
1001111	0.970	0.007	0.004
1011100	0.000	0.002	0.007
1101000	0.025	0.003	0.985
0100001	0.987	0.468	0.997
0100010	0.899	0.997	0.809
0000100	0.014	0.671	0.483
0010110	0.016	0.996	0.000

**FIGURE 5.5** Hidden node activations listed by pattern name.

Your goal is to see whether the internal representations (that is, the hidden node activation patterns) give you any clue as to the similarity structure of the patterns you have autoassociated. To do this, it would be useful to be able to group the patterns in various ways. You might wish to sort all the patterns according to the values of the first hidden node, for instance. This would group (or classify) the patterns according to how the hidden node “interpreted” them. It might be that the first hidden node discovered some feature which is present in some of the patterns but not in others, so that you find two groups of patterns. (Or perhaps not.) Similarly, you would also like to sort the patterns according to the values of the second hidden node, and the third hidden node. You can do this easily using the **Sort...** utility. This utility will sort a file, row by row. You can indicate where you want to start in the row and skip over certain columns.<sup>1</sup> Select the **Sort...** option from the **Edit** menu. The dialogue box depicted in Figure 5.6 appears. If you look at the format of your file **auto3.sort**, you will

1. You can even perform a nested sort whereby rows are first sorted by one column. Rows which match on one column then undergo sorting in a second (user specified) column.

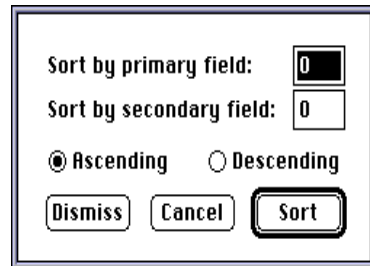


FIGURE 5.6 The Sort... dialogue box.

see that the first hidden node activation is in the second column. Thus, to **Sort** according to values of the first hidden node, you need to set the **primary field:** box in the **Sort...** dialogue to 1. We are not concerned here with performing a nested sort so leave the **secondary field:** box as 0. Click on . **tlearn** will then prompt you for a file name in which to save the sorted data. You can then **Open...** this file from the **File** menu. The lines in the resulting file will be displayed in a different order to those in **auto3.sort**. You will see that they are ordered by ascending values of the first hidden node. In a similar way, you can sort by the remaining hidden node activations. Just select the appropriate **Primary field:** in the **Sort...** dialogue.

### Exercise 5.8

- 
1. You will probably find at least one, and probably two, hidden nodes which nicely classify the input patterns. What aspects of the patterns does each hidden node seem to be attending to? Are the features in the patterns confined to single bits, or multiple bits? Do all hidden nodes succeed in finding features?
  2. Earlier we said that redundancy was a property of sets of patterns. What implications does this have for a network which was taught on a random subset of 3 of the above patterns. Would you expect similar hidden node patterns to develop? Why or why not?
-

***Exercise 5.8***

---

3. Let's say that you believe you have found a feature in the patterns which is being extracted by the network (extracted in the sense that a hidden node always is highly activated whenever a pattern with that feature is present). It would be interesting to be able to test your hypothesis, using the network you have just trained. How might you test your hypothesis? Do the test and report your results.
-

*Answers to exercises*

---

**Exercise 5.1**

- Localist representations are similar in some ways to traditional symbolic representations. Each concept is represented by a single node, which is atomic (that is, it cannot be decomposed into smaller representations). The node's semantics are assigned by the modeler and are reflected in the way the node is connected to other nodes. We can think of such nodes as hypothesis detectors. Each node's activation strength can be taken as an indicator of the strength of the concept being represented.

The advantage of localist representations is that they provide a straightforward mechanism for capturing the possibility that a system may be able to simultaneously entertain multiple propositions, each with different strength, and that the process of resolving uncertainty may be thought of as a constraint satisfaction problem in which many different pieces of information interact. Localist representations are also useful when the modeler has a priori knowledge about a system and wishes to design the model to reflect that knowledge in a straightforward way. Finally, the one-node/one-concept principle makes it relatively easy to analyze the behavior of models which employ localist representations.

The input patterns used in this exercise are localist because just one node tells you all you need to know about to identify the pattern from all the others in the training set.

**Exercise 5.2**

- Using the same logic as in Exercise 3.3 on page 63, the maximum RMS error that guarantees that all patterns have been properly encoded is given by the expression:

$$\text{RMS error} \leq \sqrt{\frac{(0.5^2)}{4}} = 0.25$$

Notice that this result is identical to the answer to Exercise 3.3. In both cases there are 4 input patterns and only one output node is supposed to be active.

**Exercise 5.3**

1. If you used the same network parameters as we suggested, you should find that the input patterns produce the hidden unit activations shown in Table 5.1. Different training parameters will yield different representa-

**TABLE 5.1** Hidden unit representations of four localist input patterns in an autoassociator.

<b>Activations</b>	
Input	Hidden
1 0 0 0	0 1
0 1 0 0	0 0
0 0 1 0	1 1
0 0 0 1	1 0

tions at the hidden unit level. The solution shown in Table 5.1 is particularly clear: Each input pattern produces a unique binary pattern across the hidden nodes. The two-dimensional hidden unit vector makes optimal use of the space of activations available to it (cf. Figure 4.14a). The different hidden unit activations permit the network to distinguish the input patterns and reproduce them at the output.

In a distributed representation, a common set of units is involved in representing many different concepts. Concepts are associated, not with individual units, but instead with the global pattern of activations across the entire ensemble. Thus, in order to know which concept is being considered at a point in time, one needs to examine the entire pattern of activation (since any single unit might have the same activation value when it participates in different patterns). For example, the activation level of the first hidden unit in **auto1** is insufficient to identify which pattern was presented at the input. It is necessary to know the activation of both hidden units.

2. The final state of the network for solving the encoding problem is depicted in Figure 5.7. Can you decipher how the network has solved the problem? For example, why is it that the second output node has a positive bias while all the others have a negative bias?

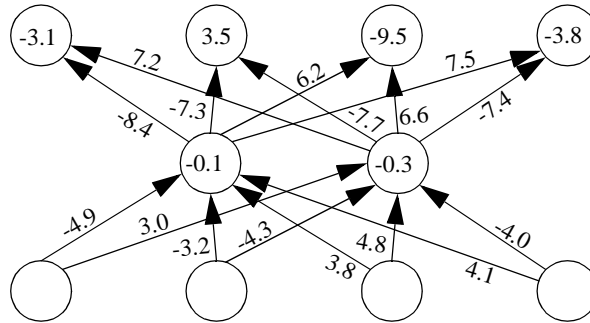


FIGURE 5.7 Network for the solution of the encoding problem in `auto1`.

#### Exercise 5.4

- At first blush, you might conclude from Exercise 5.3 that the network with 2 hidden units had used up all the distributed representations available—2 units can encode 4 binary patterns—and that more hidden units are necessary to solve the 5 pattern problem. This would be correct if the hidden units were restricted to binary activations. But, of course, they're not! They can take on any *real* value between 0 and 1. This means that there are many possible combinations of hidden unit activations that can be used to represent distinct input patterns (see *Rethinking Inmateness*, Chapter 2, Figure 2.17 and associated discussion).

So the answer is that `auto2` can solve the problem with just 2 hidden units though you might find that it finds a solution quicker with 3. Why do you think this might be?

#### Exercise 5.5

- With the suggested input pattern (0.6 0 0 0.2), the network does an excellent job of reconstructing a cleaner version of the input. However, as the noise on other input units increases in value the accuracy of completion will deteriorate. When the input is ambiguous, such as when two input units are both fully on or at 0.5, pattern completion will be particularly bad.

**Exercise 5.6**

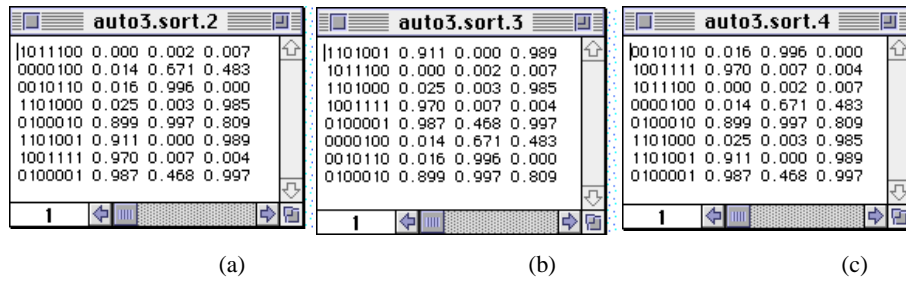
- We say that information in a pattern is redundant if one bit of the input vector can be predicted by the activity of another bit. Of course, in order to conclude that this is (or is not) the case, it is necessary to look at more than one pattern. Hence, redundancy is a property of a set of patterns, not a single pattern.

**Exercise 5.7**

1. As the patterns are displayed, it may seem that the activity of the first node divides the 8 patterns conveniently into 2 groups. But, of course, this is true of all the nodes in the patterns. Alternatively, you might decide to group the patterns in terms of the number of nodes that are activated, or whether that number is odd or even. Clearly, there are many ways in which the patterns could be categorized.
2. The network should learn to perform the task quite well given that you have used the training parameters suggested in the text.
3. Of course, your estimation of how well the network has done depends on the criteria you set for success!

**Exercise 5.8**

1. The hidden unit activations for the different input patterns are shown in Figure 5.8, sorted by first, second and third hidden units, respectively. The first hidden unit appears to categorize the patterns neatly into two distinct groups. However, it is unclear from the patterns what the basis of this categorization is. In contrast, the second hidden unit seem to be sensitive to the activity of the first and fourth input units and the third hidden unit to the activity of the second and fifth input units.
2. If you select a subset of the input patterns for the encoding problem, the nature of the task defined for the network is changed—correlations in the activity of different input units across a large set of patterns may not hold for a smaller set of patterns. The smaller set might contain the only



**FIGURE 5.8** Unit activations sorted on the first (a), second (b) and third (c) hidden units, respectively.

exceptions to the correlation in the larger set. So you shouldn't expect the hidden units to develop identical patterns of activity when trained on a subset of the patterns. Try it!

3. It is relatively easy to test any hypothesis about the cause of the hidden unit activations—just create some novel data patterns that contain only the target feature you have in mind and see if the hidden units react as you predict (using the **Testing Options...** dialogue box to select the novel data set). Of course, you can always take a close look at the weight matrix in an attempt to evaluate your hypothesis. This approach often works well for small networks, but becomes cumbersome for larger systems.