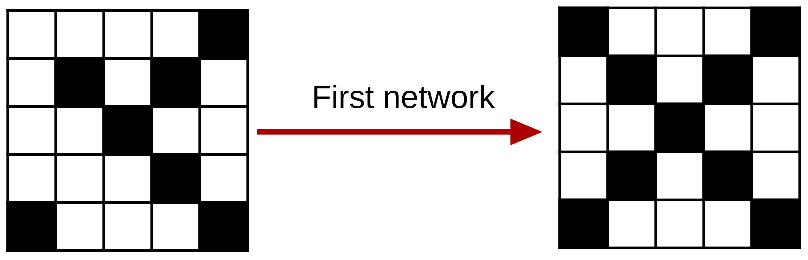
CHAPTER 11 - PATTERN CLASSIFICATION

As introduced in Chap. 10, neural networks propagate a set of neural activities λ_i^p at time t_1 to a new set λ_j^p at time t_2. The simple networks described in Chap. 10 have a number of restrictive features:
•the number of neurons at t_1 and t_2 are the same
•information is propagated only in the forward direction: there is no feedback to neurons at "earlier" times (using the word "time" rather loosely).
There is no particular need for these restrictions, and more general forms for networks are both allowed by the formalism and needed for the solution of some problems.

In Sec. 11.1, we discuss the network topology needed for pattern classification and logical operation. A generalized Hebb's rule that includes updates on synaptic thresholds is introduced in Sec. 11.2. Finally, Sec. 11.3 shows two examples of where multiple updates are necessary for the network to perform its task, or where Hebb's learning rule does not produce an accurate network.

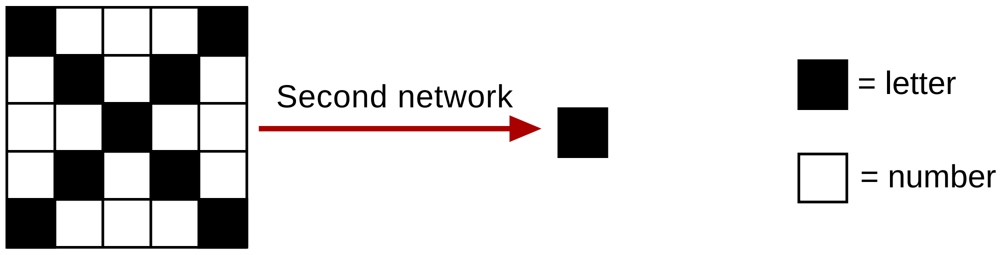
11.1 Classification and logic

In the pattern association algorithm and project of Chap. 10, the number of input neurons and output neurons is the same. But there are many situations in which this restriction is neither necessary nor desirable. For example, suppose that we want to identify a pattern as one of the 36 alphanumeric characters A, B, ... 8, 9, and then classify it as a number or a letter. We could first use a neural net à la Chap. 10 to read the input pattern (possibly with a few errors in it) and associate it with an alphanumeric pattern.



The first network maps the pattern onto one of 36 alphanumeric patterns, each with 5 x 5 = 25 elements λ_i^p (i = 1 ... 25 and p = 1 ... 36).

We could then use a second network to classify the pattern as a letter or number. This network takes an input vector with 25 elements and produces an output neuron with 1 element δ (where δ = +1 for letter and -1 for number).

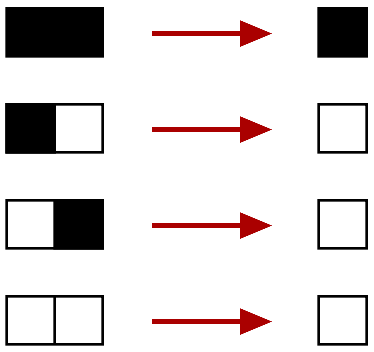


Although the classification operation reduces the number of neurons, the same generic form can be used for the neural network as in Sec. 10.2, albeit with fewer indices:

$$h = \Sigma_i w_i \lambda_i \tag{11.1}$$

$$\delta = 2 \bullet \theta(h - \phi) - 1. \tag{11.2}$$

Here, the weights have just one index, since the output neuron has only one element. Elementary logic operations also involve a reduction in the number of neurons. For example, the AND operation would have input and output patterns like (where black indicates the box is TRUE):



11.2 General Hebb's rule

Hebb's rule provides a means of determining weights and thresholds. Often, the rule is referred to as a *learning* or *training* algorithm, in that it uses known or *target* input to produce a network which hopefully has the sought-after characteristics. In its most general form, the rule reads

$$w_{ij} = N^{-1} \Sigma_p \lambda_i^p \lambda_j^p \tag{11.3}$$

$$\phi_i = -N^{-1} \Sigma_p \lambda_i^p, \tag{11.4}$$

where N is the number of elements. In the pattern association problem of Chap. 10, the thresholds were set equal to zero. If the output is simply a single element δ^p, then Eqs. (11.3) and (11.4) become

$$w_i = N^{-1} \Sigma_p \lambda_i^p \delta^p \tag{11.5}$$

$$\phi = -N^{-1} \Sigma_p \delta^p. \tag{11.6}$$

In some versions of Hebb's rule, the N^{-1} normalization factors are omitted. In other formulations, a bias b is added to the potential h, and the result $h + b$ is compared to zero in the activation function, rather than h being compared to a threshold ϕ. Clearly, $b = -\phi$, and Hebb's rule for biases (if you want to use them in place of thresholds) is

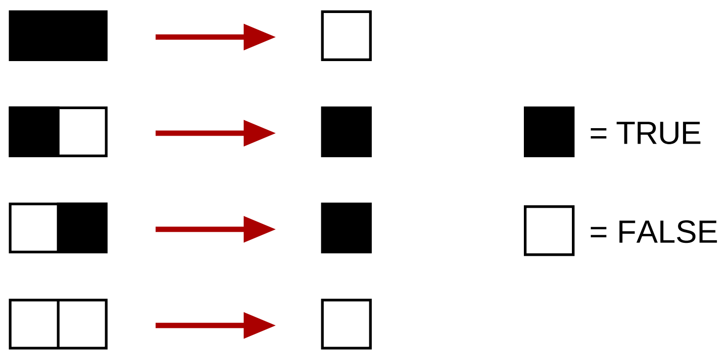
$$b = N^{-1} \Sigma_p \delta^p. \tag{11.7}$$

11.3 *Caveat emptor*

There are many situations in which the simple input/output network cannot describe a particular operation, or where the network is appropriate, but Hebb's rule does not give the correct weights and thresholds. We give an example of each.

First, we show that the input/output network does not always work. The input/output network is also called a *single layer* network, since the weights act only once. The classic example is the exclusive-OR operation, also known as XOR. In this operation, the output is true only if one, and only one, of the inputs is true.

Diagrammatically,



Representing this situation by a single layer network, there are three unknowns (w_1, w_2 and ϕ) and four equations. It turns out that the unknowns are overspecified. For each of the four logic operations in the diagram, the corresponding equations are

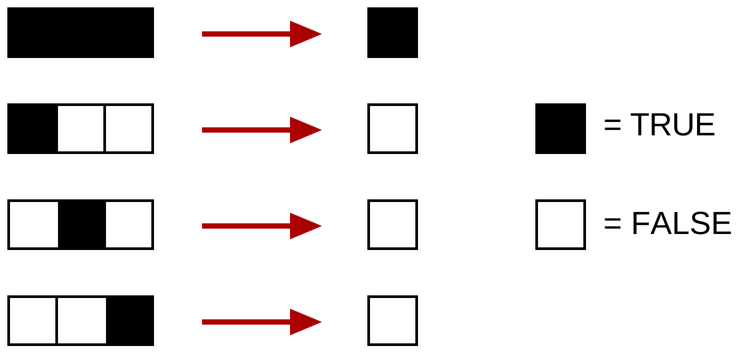
$$w_1 + w_2 < \phi \tag{11.8a}$$
$$w_1 - w_2 > \phi \tag{11.8b}$$
$$-w_1 + w_2 > \phi \tag{11.8c}$$
$$-w_1 - w_2 < \phi \tag{11.8d}$$

Note that Eqs. (11.8a) and (11.8d) form a couplet, as do (11.8b) and (11.8c). A few minutes of trial-and-error effort shows that the equations cannot be satisfied. Even with liberal interpretation of the step function at $x = 0$, the only solution to the equations is

$$w_1 = w_2 = \phi = 0. \tag{11.9}$$

This set of weights cannot classify any patterns. Neural networks can be made to represent XOR, but a second layer of weights must be added.

An example of how Hebb's rule fails to yield the correct network, even where the network has a proper solution, is the following classification problem:



According to Hebb's rule, Eq. (11.5) and (11.6):

$$w_1 = w_2 = w_3 = (1 - 1 + 1 - 1) / 3 = 0 \tag{11.10a}$$

$$\phi = - (1 - 1 - 1 - 1) / 3 = +2/3. \tag{11.10b}$$

Application of this set of weights to the first pattern of the set shows that the weights do not work. Does this problem originate with the assumptions behind the network itself, or with Hebb's rule? The problem is Hebb's rule, since the solution

$$w_1 = w_2 = w_3 = 1/3 \tag{11.11a}$$

$$\phi = +2/3 \tag{11.11b}$$

correctly reproduces the training data.

References

L. Fausett, *Fundamentals of Neural Networks* (Prentice-Hall, Englewood Cliffs, NJ, 1994) Chaps. 2-3.

B. Muller and J. Reinhardt, *Neural Networks: an Introduction* (Springer-Verlag, Berlin, 1990), Chaps. 1-4.

11.4 Project 11 - Tick-tack-toe

Suppose that you are part of a design team building a game droid - a robot that can play simple games like tick-tack-toe, or bingo. The games are to be played on paper, perhaps even in a bingo hall, and the droid must be able to write, and to recognize writing. Further, the droid must be able to categorize and respond to patterns. Your task on the tick-tack-toe design team is to implement neural networks that will:

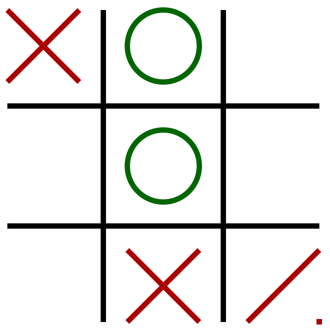
•recognize handwritten X's and O's (or noughts and crosses) on the game card

•categorize multiple X/O patterns as part of the droid's strategy.

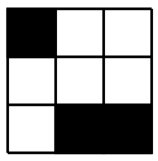
You have already implemented the X/O pattern association in the project of Chap. 10. In this section, you develop a classification algorithm.

Physical system

The tick-tack-toe game card has the usual 3x3 form

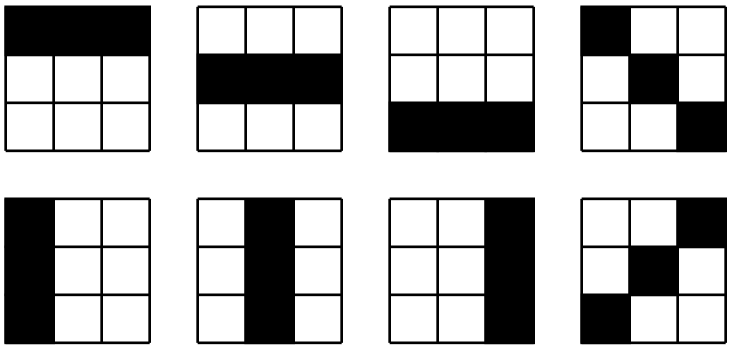


Each element of the grid assumes one of three values (X, O, blank), which is actually a little more than we wish to consider. So, for this project, we will make the system binary, such that every square in the 3x3 grid is either black or white.



The task is to classify patterns as winning or losing for either black or white. Since the algorithm is the same for either color, then you need only code up the algorithm for the black squares.

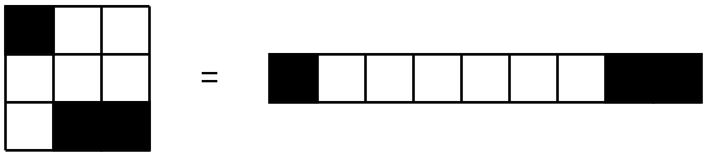
There are 2^9 black-and-white patterns, about 55% of which represent winning combinations for either black or white. There are 8 fundamental patterns that represent a win:



Obviously, more black squares can be added to any of the fundamental patterns, and the resulting pattern will still "win". In six of the eight patterns shown, both black and white have complete rows and are therefore "winning" patterns.

Simulation parameters

Although the patterns are 3x3 arrays, it is more convenient to convert them to 9-component vectors (see Chap. 10):



The patterns whose win/lose characteristics are known are denoted by λ_i^p, where *i* is the 9-component vector index and *p* is the label of the pattern. The win/lose characteristic of the pattern (the *target*) is denoted by t^p. For ease of using Hebb's rule for network training, the bipolar convention +1 / -1 should be used for both the nine elements of λ_i and the single value of *t* for each pattern.

The network must take a pattern L_i and determine its win/lose characteristic *H* through the conventional linear relation for the synaptic potential

$$h = \sum_i w_i L_i \qquad (11.12)$$

$$H = \mathrm{sgn}(h - \phi), \tag{11.13}$$

where ϕ is an activation threshold. Hebb's learning rule is used to determine the 9 weights and one threshold through

$$w_i = \Sigma_p \lambda_i^p t^p \tag{11.14}$$

$$\phi = - \Sigma_p t^p. \tag{11.15}$$

This set of learning rules does **NOT** give a set of weights that are 100% accurate, but it's the best that we can do in the limited time available in this course.

Code

The code consists of many segments of a few lines each. One way of organizing and writing the code would be something like:

1. Write a routine (**test_it**) that will take a 9-component pattern λ_i^p and determine its win/lose characteristic t^p for one colour, black or white. Use the +1 / -1 convention for all elements.

2. Write a routine (**train_it**) that generates all 512 possible patterns λ_i^p. As each pattern is generated, call **test_it** to find t^p. Then, use both λ_i^p and t^p to find w_i and ϕ by Eqs. (11.14) and (11.15).

3. Write a routine (**net_it**) to take an unknown pattern L_i and predict its corresponding value of t.

4. Generate random patterns, and compute their corresponding t-values by
•**test_it** to determine t "exactly"
•**net_it** to predict t from the net.

5. Record the accuracy of the net.

Report

Your report should include:
•a statement of the problem to be solved
•Hebb's learning rule
•an outline of your code
•an analysis of your code's accuracy
•a copy of your code.

Demo code

The neural network in the demo code was prepared using the steps above in the *Code* description. You can manipulate the occupied elements in the 3x3 grid, and the code tells you whether the "white" player has a winning pattern. While you will find it tiresome to enter all 512 patterns to determine the accuracy of the network, it is not difficult to find combinations that the neural network predicts incorrectly.