Neural Networks

Lab Assignment 3: The Hopfield-network

1 Introduction

This week's lab assignment is about the Hopfield network. This network implements an associative memory by storing the relationships between different features of some training data. While training the network, the weights between the neurons that often fire simultaneously are strengthened, whereas the weights between neurons that rarely fire simultaneously are weakened. This is called Hebbian learning, as Donald Hebb described this principle first for biologic neurons. The principle of Hebbian learning is nicely captured in the next phrase:

Cells that fire together, wire together.
When cells fire apart, their wires depart.

After applying Hebbian learning, the network will be able to remove noise from patterns by using its associative memory. The recovery of the pattern can be considered as the minimization of an energy function. Chapter 7 of the book covers everything you need to know for this lab assignment.

2 Theory questions (2 pt.)

a) Draw a Hopfield network with 3 neurons and indicate which neurons are active.

b) Is the Hopfield network that you drew in an equilibrium? Explain why.

c) What are two key features of the weights in a Hopfield network?

d) How do you compute the activation of a neuron in a Hopfield network?

e) Which type of activation function is used in a Hopfield network?

f) Which kind of neurons does a Hopfield network contain when you consider your answers to (e) and (f)?

3 Implementing a Hopfield network in MATLAB

Now we are going to implement a Hopfield network in MATLAB. On Nestor you can find a zip-file `NNLab3.zip`. The source file `hopfield.m` can be found here along with the data file `m.dat`, `a.dat`, `s.dat`, `t.dat`, `e.dat` and `r.dat`. The data files contain 5x5 matrices that look like letters of the word *master*.

3.1 Training (2 pt.)

For the training we will use one-shot Hebbian learning. This means that we process all training data at once. Compute the weight matrix W in which the element at i, j is given by:

$$w_{ij} = \sum_{p=1}^{P} v_i^p v_j^p$$

where P is the number of training patterns and v_i^p is the i-th value of the p-th training pattern. This formula can be found on page 106 in the book. You could compute W as a multiplication of two matrices, but you can also use a `for`-loop if you find that more comprehensible. The training data are in `vector_data`.

Now change the weight matrix such that the neurons are not connected to themselves anymore.

3.2 Updating the state (1 pt.)

Now that we have trained the network it is time to update the state. As you can see we process all training patterns at once after which the network will update itself for a certain number of epochs.

Implement the synchronous network update. The activation of the network after the previous update (or after the original input) is stored in the vector `activation`.

Implement the activation function and make sure that the activation of each neuron is the corresponding entry of `activation`. For further details, see page 97/98 of the book.

4 Questions on the Hopfield implementation (2 pt.)

Now we have finished the implementation of the network. Vary the number of epochs and number of patterns `n_epochs` and `n_examples`. Answer the following questions.

a) Explain why repetitively applying Hebb's learning rule can be captured by the following formula

$$w_{ij} = \sum_{p=1}^{P} v_i^p v_j^p$$

b) Omit the normalization of the weights and rerun the algorithm. What happens to the performance of the network? How can you explain this?

c) The number of different patterns that can be stored in a Hopfield network is dependent on the number of neurons. Provide the formula with which you can determine the number of patterns that a Hopfield network can store given the number of neurons.

d) What is the theoretical number of patterns that a Hopfield network using 25 neurons can store?

e) Vary `n_examples`. How many different patterns can be stored properly by the network? Why is this different from what you have found in (d)?

5 Noise and spurious states (2 pt.)

Neural networks are known for their ability to generalize. Like many other neural networks, the Hopfield network is able to deal with noise and can even be used to 'repair' a distorted pattern.

Vary the noise level (e.g. try 30, 40 or 50 percent) and answer the following questions. Use 3 different patterns.

a) Explain using your own words how the network is able to recover the patterns from the noisy input. Your answer should at least involve `energy` and `weights`.

b) Occasionally, for higher noise levels ($30 - 35\%$) some peculiar end states might occur. Explain why these states are only observed with synchronous updates and not with asynchronous updates. If you cannot find any deviant end states then you can increase the chance of observing one by using more patterns and increasing the amount of epochs.

c) Set `invert` in the beginning of the code to `true`. Explain what happens to the input. Does anything change for the training of the network?

d) Explain why the inverted patterns are also stored by the network.