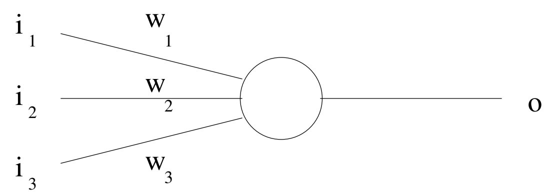

Figure 1: An artificial neuron as used in a Hopfield network.

Hopfield networks

1 Introduction

Hopfield networks are constructed from artificial neurons (see Fig. 1). These artificial neurons have N inputs. With each input i there is a weight w_i associated. They also have an output. The state of the output is maintained, until the neuron is updated. Updating the neuron entails the following operations:

- The value of each input, x_i is determined and the weighted sum of all inputs, $\sum_i w_i x_i$ is calculated.
- The output state of the neuron is set to +1 if the weighted input sum is larger or equal to 0. It is set to -1 if the weighted input sum is smaller than 0.
- A neuron retains its output state until it is updated again.

Written as a formula:

$$o = \begin{cases} 1 & : \sum_i w_i x_i \geq 0 \\ -1 & : \sum_i w_i x_i < 0 \end{cases}$$

A Hopfield network is a network of N such artificial neurons, which are fully connected. The connection weight from neuron j to neuron i is given by a number w_{ij}. The collection of all such numbers is represented by the weight matrix W, whose components are w_{ij}.

Now given the weight matrix and the updating rule for neurons the dynamics of the network is defined if we tell in which order we update the neurons. There are two ways of updating them:

- Asynchronous: one picks one neuron, calculates the weighted input sum and updates immediately. This can be done in a fixed order, or neurons can be picked at random, which is called *asynchronous random updating*.
- Synchronous: the weighted input sums of all neurons are calculated without updating the neurons. Then all neurons are set to their new value, according to the value of their weighted input sum. The lecture slides contain an explicit example of synchronous updating.

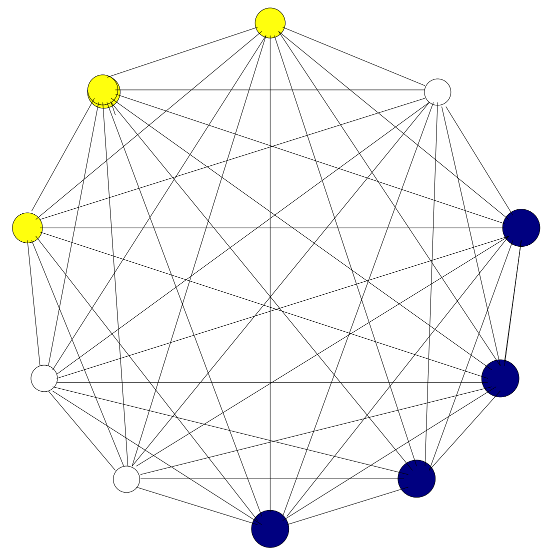

Figure 2: A Hopfield network as an autoassociator. One enters a pattern in blue nodes and let the network evolve. After a while one reads out the yellow nodes. The memory of the Hopfield network associates the yellow node pattern with the blue node pattern.

2 Use of the Hopfield network

The way in which the Hopfield network is used is as follows. A pattern is entered in the network by setting all nodes to a specific value, or by setting only part of the nodes. The network is then subject to a number of iterations using asynchronous or synchronous updating. This is stopped after a while. The network neurons are then read out to see which pattern is in the network.

The idea behind the Hopfield network is that patterns are stored in the weight matrix. The input must contain part of these patterns. The dynamics of the network then retrieve the patterns stored in the weight matrix. This is called Content Addressable Memory (CAM). The network can also be used for auto-association. The patterns that are stored in the network are divided in two parts: cue and association (see Fig. 2). By entering the cue into the network, the entire pattern, which is stored in the weight matrix, is retrieved. In this way the network restores the association that belongs to a given cue.

The stage is now almost set for the Hopfield network, we must only decide how we determine the weight matrix. We will do that in the next section, but in general we always impose two conditions on the weight matrix:

- symmetry: $w_{ij} = w_{ji}$
- no self connections: $w_{ii} = 0$

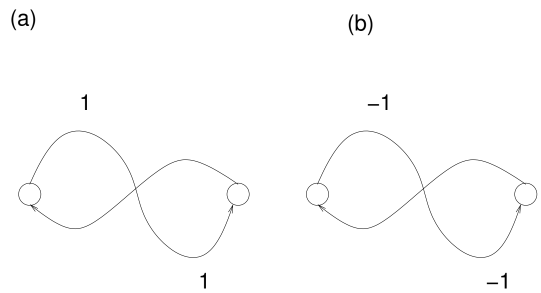

Figure 3: A two-neuron network. There are just two options for the weight matrix: $w_{ij} = 1$ (A) or $w_{ij} = -1$. If the weight is 1, there are two stable states under synchronous updating $\{+1, +1\}$, or $\{-1, -1\}$. For a weight of -1, the stable states will be $\{-1, +1, \}$ or $(+1, -1\}$ depending on initial conditions. Under synchrononous updating the states are oscillatory.

It turns out that we can guarantee that the network converges under asynchronous updating when we use these conditions.

3 Training the network

3.1 A simple example

Consider the two nodes in Fig. 3 A. Depending on which values they contain initially they will reach end states {+1, +1} or {-1,-1} under asynchronous updating. The nodes in Fig. 3 B on the other hand will reach end states {-1, +1} and

Exercise: Check this!

Exercise: Check that under synchonous updating there are no stable states in the network.

This simple example illustrates two important aspects of the Hopfield network: the steady final state is determined by the value of the weight and the network is 'sign-blind'; it depends on the initial conditions which final state will be reached, {+1,+1} or {-1, -1} for 3 A. It also shows that synchronous updating can lead to oscillatory states, something which is not true for asynchronous updating (see section 4.2).

4 Setting the weight matrix

4.1 A single pattern

Consider two neurons. If the weight between them is positive, then they will tend to drive each other in the same direction: this is clear from the two-neuron network in the example. It is also true in larger networks: suppose we have

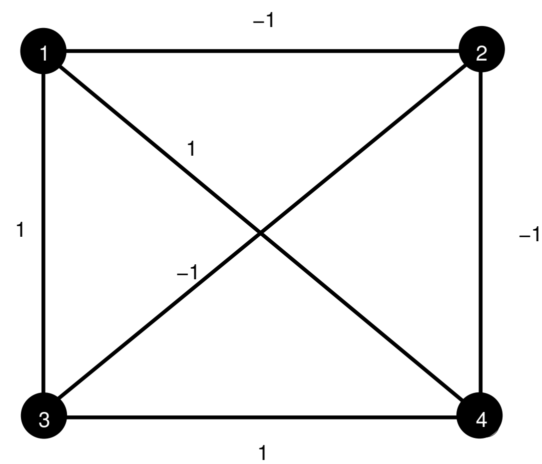

Figure 4: A four-node network to store pattern (1, -1, 1, 1). Because $w_{ij} = w_{ji}$ we have reprsented the two weights by a single link.

neuron i connected to neuron j with a weight of $+1$, then the contribution of neuron i to the weighted input sum of neuron j is positive if $x_i = 1$ and negative if $x_= -1$. In other words neuron, i tries to drive neuron j to the same value as it has currently. If the connection weights between them is negative, however, neuron i will try to drive neuron j to the opposite value! This inspires the choice of the Hebb rule: given a network of N nodes and faced with a pattern $\vec{x} = (x_1, ..., x_N)$ that we want to store in the network, we chose the values of the weight matrix as follows:

$$w_{ij} = x_i x_j \tag{1}$$

To see how this works out in a network of four nodes, consider Fig. 4. Note that weights are optimal for this example in the sense that if the pattern (1, -1, 1, 1) is present in the network, each input node receives the maximal or minimal (in the case of neuron 2) input some possible!

We can actually prove this: suppose the pattern $\vec{x}$, which has been used to train the network, is present in the network and we update neuron i, what will happen? The weighted input sum of a node is often denoted by $h_i \equiv \sum_j w_{ij} x_j$, which is called the *local field*. If w_{ij} was chosen to store this very same pattern, we can calculate h_i:

$$h_i = \sum_j w_{ij} x_i x_j = \sum_{j=1}^{3} x_i x_j x_j = x_i + x_i + x_i = 3x_i \tag{2}$$

This is true for all nodes i. So all other three nodes (self connections are forbidden!) in the network give a summed contribution of $3x_i$. This means that x_i will never change value, the pattern is stable!

What happens if instead of $\vec{x}$ we have another pattern in the network, $\vec{v} = -\vec{x}$? What happens now if we try to update it? Again for the local field h_i we find:

$$h_i = \sum_j w_{ij} v_j = -\sum_j w_{ij} x_j = -\sum_j x_i x_j x_j = -\sum_{j=1}^{3} x_i = -3x_i = 3v_i \qquad (3)$$

Again, this pattern is maximally stable: the Hopfield network is 'sign-blind'.

4.2 Energy

We can define an energy for each node:

$$E_i = -\frac{1}{2} h_i x_i \qquad (4)$$

Note that the energy is positive if the sign of h_i and x_i is different! An update of node i will result in a sign change in this case because the local field h_i has a different sign. The updating will change the energy form a positive number to a negative number. This corresponds to the intuitive idea that stable states have a low energy. We can define an energy for the entire network:

$$E(\vec{x}) = \sum_i E_i = -\sum_i \frac{1}{2} h_i x_i = -\frac{1}{2} \sum_{ij} w_{ij} x_i x_j \qquad (5)$$

Again, it is clear that for the pattern that has been used to train the weight matrix the energy is minimal. In this case:

$$E = -\frac{1}{2} \sum_{ij} w_{ij} x_i x_j = -\frac{1}{2} \sum_{ij} x_i x_j x_i x_j = -\frac{1}{2} \sum_{ij} 1 = -\frac{1}{2}(N-1)^2 \qquad (6)$$

5 Stability of a single pattern

So far we have looked at the situation where the same patterns was in the network that was used to train the network. Now let us assume that another pattern is in the network. It is pattern $\vec{y}$, which is the same as pattern $\vec{x}$ except for three nodes. The network, as usual, has N nodes. No we are updating a node i of the network. What is the local field? It is given by:

$$h_i = \sum_j w_{ij} y_j = \sum_j x_i x_j y_j \qquad (7)$$

We can split this sum in 3 nodes, which have an opposite value of $\vec{x}$ and $N-3$ nodes which have the same value:

$$h_i = \sum_{j=1}^{N-3} x_i x_j x_j + \sum_{j=1}^{3} x_i x_j \cdot -x_j = (N-3)x_i - 3x_i = (N-6)x_i$$

For $N > 6$ all the local fields point the direction of x_i, the pattern that was used to define the weight matrix!. So updating will result in no change (if the value of node i is equal to x_i) or an update (if the value of node i is equal to -x_i). So, the pattern that is stored into the network is stable: up to half of the nodes can be inverted and the network will still reproduce $\vec{x}$.
Exercise: To which pattern will the network converge if more than half the nodes are inverted?

There is also another way to look at this. The pattern that was used to train the network is an absolute energy minimum. This follows from equation 6. One can show the following important proposition, which is true for *any symmetric weight matrix*:

Proposition 1

The energy of a Hopfield network can only decrease or stay the same. If an update cause a neuron to change sign, then the energy will decrease, otherwise it will stay the same.
The condition that the energy of a Hopfield network can only decrease only rests on the condition that the weight matrix is symmetric. This is true for the Heb rule and also the generalised Hebb rule (see below). The undergraduates need not know this proof, but the postgraduates must be able to reproduce it. We will proof proposition 1 in section A.

Since there is only one absolute minimum in a Hopfield network that has been trained with a single pattern (equation 6 is proof of that) and the minimum is reached when the training pattern (or its negative inverse; the same pattern with multiplied by -1) is in the network, the network must converge to the trained pattern (or its negative inverse).

6 The Hebb rule and the generalised Hebb rule

Instead of setting the weights by equation 1 we usually use:

$$w_{ij} = \frac{1}{N} x_i x_j \tag{8}$$

Exercise: Why does this not make a difference for the dynamics of the network?

How do we store more than one single pattern in the network? Let us assume we have two patterns $\vec{x^{(1)}}$ and $\vec{x^{(2)}}$ that we want to store. The trick is to calculate the weight matrix for pattern one, as if the other pattern dies not exist:

$$w_{ij}^{(1)} = \frac{1}{N} x_i^{(1)} x_j^{(1)}$$

and

$$w_{ij}^{2} = \frac{1}{N} x_i^{(2)} x_j^{(2)}$$

and the to add them:

$$w_{ij} = w_{ij}^{(1)} + w_{ij}^{(2)}$$

for p patterns the procedure is the same and this leads to the generalised Hebb rule:

$$w_{ij} = \frac{1}{N} \sum_{k=1}^{p} x_i^{(k)} x_j^{(k)}$$

where x_i^k is the value of node i in pattern k.

7 Energy and the generalised Hebb rule

The two weight matrices interfere with each other. The trained patterns are no longer absolute energy minima. One can show, however, that they are still local energy minima, if the network is large and the patterns do not resemble each other to closely (are uncorreleated). The network is $(p \ll N)$, where p is the number of patterns used to train the network and N the number of nodes. This means that if you enter a pattern in the network and then let network dynamics take over that you will still end up in an energy minimum. The generalised Hebb rule still satisfies the condition of Prosition 1, so network dynamics guarantees that the network will converge towards an energy minimum and therefore to a steady final state. And it is highly likely that such a minimum will correspond to one of the training patterns. It is not guaranteed, however.

7.1 Spurious minima

There are so-called *spurious minima*. These are patterns which are local minima, but do not correspond to a training pattern. If more than three patterns are stored in the network, one can show, for example, that asymmetric 3-mixtures are also local minima. What are those? Given three training patterns, one can create a fourth one. For each node, the value is defined by the majority values of the other node: if the other patterns at node i have the values -1, -1, 1, respecively, the value of node i in the 3-mixture will be -1. Example, suppose that in a 10-node network, there are three training patterns:

$$\begin{aligned} x^{(1)} &= (1, -1, 1, -1, 1, -1, 1, -1, 1, -1) \\ x^{(2)} &= (1, -1, -1, -1, 1, 1, 1, -1, -1, -1) \\ x^{(3)} &= (1, 1, 1, 1, 1, -1, -1, -1, -1, -1) \end{aligned} \tag{9}$$

then the 3-mixture is:

$$(1, -1, 1, -1, 1, -1, 1, -1, -1, -1)$$

The redeeming quality of Hopfield networks is that the so-called 'basin of attraction' is much smaller for 3-mixtures than for training patterns. So, although one may sometimes end up in a mixture of training patterns, as the demonstration showed, one is much more likely to end up in a training pattern, provided p llN and the training patterns are not too strongly correlated.

A Proof of Proposition 1

Optional for undergraduates; required for postgraduates In general, when a pattern $\vec{x}$ is in the network, its energy is given by:

$$E(\vec{x}) = -\frac{1}{2}\sum_{ij} x_i x_j$$

About w_{ij} we will only assume that it is symmetric, it does not necessarily have the Hebb form! This is a large sum of terms in a three-node network, for example:

$$E(\vec{x}) = -\frac{1}{2}(w_{11}x_1x_1 + w_{12}x_1x_2 + w_{13}x_1x_3 + w21x_2x_1 + w_{22}x_2x_2 + w_{23}x_2x_3 + w_{31}x_3x_1 + w_{32}x_3x_2 + w_{33}x_3x_3)$$

Now suppose we update one single neuron x_p, then it will either retain the same value, or it will change. Proposition 1 says that if it changes, the energy of the network will be lower. This can be shown by explicit calculation. Let $E(t)$ be the energy before the update, given pattern $\vec{x}$ and $E(t+1)$ the enrgy after update. Then

$$E(t) = -\frac{1}{2}\sum w_{ij}x_i x_j = -\frac{1}{2}\sum_{ij, i\neq p, j \neq p} w_{ij}x_i x_j - \frac{1}{2}\sum_j w_{pj}x_p x_j - \frac{1}{2}\sum_i w_{ip}x_i x_p$$

This is just a rewriting of the energy sum and splitting off those terms which depend on x_p. Let the new value of x_p be x_p^*. Then for $E(t+1)$ we find:

$$E(t) = -\frac{1}{2}\sum w_{ij}x_i x_j = -\frac{1}{2}\sum_{ij, i\neq p, j \neq p} w_{ij}x_i x_j - \frac{1}{2}\sum_j w_{pj}x_p^* x_j - \frac{1}{2}\sum_i w_{ip}x_i x_p^*$$

We can then calculate

$$\Delta E = E(t+1) - E(t)$$

. Since this only involves the terms which depend on x_p, x_p^*, we can write

$$\Delta E = -\frac{1}{2}\sum_j w_{pj}x_p^* x_j - \frac{1}{2}\sum_i w_{ip}x_i x_p^* + \frac{1}{2}\sum_j w_{pj}x_p x_j + \frac{1}{2}\sum_i w_{ip}x_i x_p$$

Using the symmetry of the weight matrix, one finds:

$$\Delta E = \sum_i w_{pi}x_i(x_p - x_p^*) \tag{10}$$

Now, there are two possibilities for an update:

$$x_p : -1 \to 1$$

In this case $(x_p - x_p^*) = -2$, but for such an update to take place $\sum_i w_{pi} x_i > 0$. So $\Delta E < 0$. The other possibility is

$$x_p : 1 \rightarrow -1$$

. It is easy to see that then also $\Delta E < 0$. This concludes the proof.