Neural Networks

Assignment 3 - The Hopfield network

George Petridis (s3001393)
Jussi Boersma (s2779153)

March 16, 2017

2 Theory questions

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

In Figure 1 we have drawn a Hopfield network with 3 neurons. The active neurons are labeled with the red 'a'.

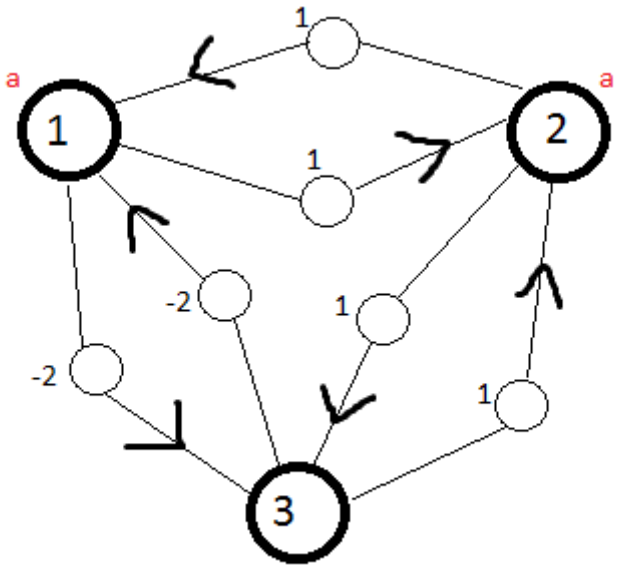

Figure 1: A Hopfield network with 3 neurons.

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

The Hopfield network that we drew is in an equilibrium, because updating the network can only result in the same amount of energy.

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

Two key features of the weights in a Hopfield network are:

- Weights between nodes are always the same (symmetrical).
- Weights from a node to itself are always 0.

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

The activation of a neuron in a Hopfield network is computed by a vector multiplication between the neuron vector $\vec{x}$ and the the weight vector $\vec{w}$.
$\vec{x}$ is as big as the number of neurons and takes the value 1 for an active neuron and 0 for an inactive neuron, while $\vec{w}$ consists of the weights between neurons.

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

In the Hopfield network a binary step function is used to calculate the activation of a neuron. The neuron fires when its activation is bigger than or equal to 0, while the opposite happens if the activation is smaller than 0.

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

Considering how the Hopfield network activation is computed and the type of activation function that is used in such a network, we can say that a Hopfield network contains TLU neurons, where the activation is computed in the same way and with the same function.

3 Implementing a Hopfield network in MATLAB

3.1 Training

We computed W as a multiplication of two matrices:

```
% The result should be a matrix dimensions: size_examples * size_examples
% Each entry should contain a sum of n_examples
weights = transpose(vector_data) * vector_data;
```

We then changed the weight matrix so as for the neurons to not be connected to themselves anymore. This was achieved by setting all the values in the diagonal of the weight matrix to 0, as follows:

```
% A hopfield neuron is not connected to itself. The diagonal of the matrix
% should be zero.
weights(1:size_examples+1:size_examples*size_examples) = 0;
```

3.2 Updating the state

We computed the new activation as follows:

```
% Compute the new activation
activation = weights * activation;
```

And then we applied the activation function:

```
% Apply the activation function
activation = (activation >= 0);
activation = (activation * 2) - 1;
```

4 Questions on the Hopfield implementation

4.a Explain why repetitively applying Hebb's learning rule can be captured by the following formula $\mathbf{w}_{ij} = \sum u_i^p u_i^j$

Repetitively applying Hebb's learning rule can be captured by the above formula, because when both the terms have weights of the same sign, the outcome of the sum is positive, thus the weight will be positive as well. In other words, terms like 1*1 and -1*-1 both result in positive values for the weight, whereas terms of opposite sign like 1*-1 and -1*1 result in a negative weight. This holds with Hebb's learning rule:
Cells that fire together, wire together. When cells fire apart, their wires depart.

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

By omitting the normalization of the weights and rerunning the algorithm, we saw that the network's number of errors increased. In fact we noticed an increase of twice as many errors without normalization, than those with normalization. This can be explained by the fact that the activation is using values between -1 and 1 for its computation, therefore by normalizing the weights they become more realistic predictors with regard to the computed activation.

4.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.

The formula for the number of patterns (m) that a Hopfield network can store given the number of neurons (N) is:

$$m < \frac{N}{2 * ln(N)}$$

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

By using the above formula we get the following theoretical number of patterns using 25 neurons:

$$\frac{25}{2 * ln(25)} = 3.88 \text{ Patterns}$$

4.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)?

After a lot of failed attempts using 4 patterns, our conclusion is that the maximum amount of different patterns that can be stored is 3. This is because theoretically 4 patterns is impossible, as it approaches the solution, but doesn't perfectly reach it. Since only integer patterns can be used in our algorithm, the number that is theoretically found should be rounded off downwards.

5 Noise and spurious states

5.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.

The weights in the network are based on the ratio between the pixels in every training image on an (i, j) location. Using these weights, the algorithm adjusts the noised input with its weights for a set number of epochs. This way, the network evolves towards the lowest possible energy. It is not necessarily possible to reach this energy with the number of epochs that we use, therefore it will not always find an optimal solution.

5.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.

Figure 2 shows an example of peculiar end states that can occur for high noise levels.
For both the letters 'M' and 'T', the network returns images that do not look like one of the goal patterns. This does not happen when using asynchronous updating, because the network updates the individual neurons one by one rather than all at once. This results in the nodes that are evaluated later in the sequence to profit of the improvements made in the same epoch and the network improves continuously over time.

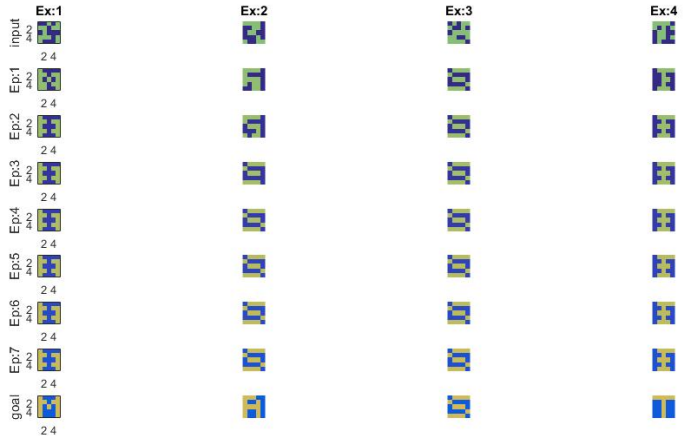

Figure 2: Peculiar end states that occur with high noise levels.

5.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?

When we set invert to true, the input is inverted after the noise is added. With higher noise it seems that the network's training results in better classification than without inversion.

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

The inverted patterns are also stored by the network, because the negative values in the weights already relate to the inverse input, just as the positive values relate to the regular input.