Artificial Neural Networks and Deep Architectures,
DD2437

Lab assignment 3

Hopfield networks

1 Objectives and scope

This exercise is predominantly concerned with Hopfield networks and associative memory. When you are finished you should be able to:

- explain the principles underlying the operation and functionality of autoassociative networks,
- train the Hopfield network,
- explain the attractor dynamics of Hopfield networks the concept of energy function,
- demonstrate how autoassociative networks can do pattern completion and noise reduction,
- investgate the question of storage capacity and explain features that help increase it in associative memories.

Most of the operations of the Hopfield-type of networks can be seen as vector-matrix operations.

First, we will look at some simple networks using the Hebbian learning principle, which is often employed in recurrent networks. You will construct an autoassociative memory of the Hopfield type, and explore its capabilities, capacity and limitations. In most of the tasks, you will be asked to study the dynamics and analyse its origins (potentially explain different behaviours you observe).

2 Background

A neural network is called *recurrent* if it contains connections allowing output signals to enter again as input signals. They are in some sense more general than feedforward networks: a feedforward network can be seen as a special case of a recurrent network where the weights of all non-forward connections are set to zero.

One of the most important applications for recurrent networks is *associative memory*, storing information as dynamically stable configurations. Given a

noisy or partial pattern, the network can recall the original version it has been trained on (*autoassociative memory*) or another learned pattern (*heteroassociative memory*).

The most well-known recurrent network is the *Hopfield network*, which is a fully connected autoassociative memory network with two-state (bipolar -1,1 or binary 0,1) neurons. One reason that the Hopfield network has been so well studied is that it is possible to analyse it using methods from statistical mechanics, enabling exact calculation of its storage capacity, convergence properties and stability. In this assignment you will solely focus on the most popular discrete version of the Hopfield network with *Hebbian learning rule* and both synchronous (simultaneous) and asynchronous (sequential) update during recall.

2.1 Hebbian learning

The basic idea suggested by Donald Hebb can be briefly summarised as follows: assume that we have a set of neurons connected to each other through synapses. When the neurons are stimulated with some input pattern, correlated activity causes the respective synapses to grow, thereby strengthening the connections between correlated pairs of neurons. This will facilitate co-activations within the emerging groups of neurons, referred to as cell assemblies by Hebb. One of the implications of these co-activating populations is a functional feature of pattern completion. In particular, when only part of the memory pattern (partial cue or stimulus) is used to cue a memory network, stimulated neurons will likely activate other neurons in their assembly through strengthened connections (by means of earlier Hebbian learning) (see figure 1).

If two neurons are on the other hand anti-correlated, the synapses between them get weakened or even become inhibitory. This way a pattern of activity precludes other learned patterns of activity. This makes the network noise resistant, since a neuron not belonging to the current pattern of activation will be inhibited by the active neurons.

This form of learning is called Hebbian learning, and is one of the most used non-supervised forms of learning in neural networks. Bear in mind its local nature.

To efficiently implement Hebbian learning, the calculations should be performed in matrix notation. The units can be indexed by i and the desired activationsnof the training patterns are vectors $\bar{x}^\mu$ with components x_i^μ, where μ is the number of the patterns. We could use for instance 0 and 1 for the activities but the calculations become easier if we choose -1 and 1 (bipolar units).

To measure the correlated activities we can use the outer product $W = \bar{x}^T \bar{x}$ of the activity vectors we intend to learn. If the components x_i and x_j are correlated w_{ij} will be positive, and if they are anti-correlated w_{ij} will be negative. Note that W is a symmetric matrix; each pair of units will be connected to each other with the same strength.

The coefficients for the weight matrix can then be written as:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{P} x_i^\mu x_j^\mu$$

where μ is index within a set of patterns, P is the number of patterns, and N is the number of units.

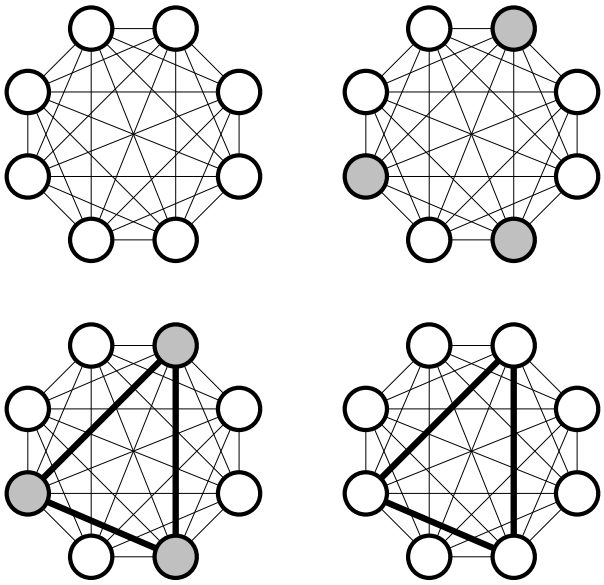

Figure 1: A simple Hebbian fully connected auto-associative network. When three units are activated by a stimulus, their mutual connections are strengthened. The next time any of them gets activated, it will tend to activate the other neuron.

2.2 Hopfield network recall

To recall a pattern of activation $\bar{x}$ in this network we can use the following update rule:

$$x_i \leftarrow \text{sign}\left(\sum_j w_{ij} x_j\right)$$

where

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

In fact, this variant of the Hopfield network, where all states are updated synchronously, is less popular and goes under the nickname of the *Little model*. In the original Hopfield model the states are updated one at a time, allowing each to be influenced by other states that might have changed sign in the previous update steps. The choice of asynchronous or synchronous update has some effects on the convergence properties (see section 3.3). The Little model is very easy to implement relying on matrix operations.

Enter three memory patterns

```
x1=[-1 -1  1 -1  1 -1 -1  1]
x2=[-1 -1 -1 -1 -1  1 -1 -1]
x3=[-1  1  1 -1 -1  1 -1  1]
```

The next step is to calculate a weight matrix. In fact, you do not need to scale the weight matrix with the $\frac{1}{N}$ factor (the scaling becomes important only if there is a bias term, a transfer function with a more complicated shape or if we look at the energy as in section 3.3).

If the network has succeeded in storing the patterns `x1`, `x2` and `x3`, they should all be *fixed points* when you apply the update rule. This simply means that when you apply the update rule to these patterns you should receive the same pattern back (content addresable memory).

- Check if the network was able to store all three patterns.

3 Tasks and questions

3.1 Convergence and attractors

Can the memory recall the stored patterns from distorted inputs patterns? Define a few new patters which are distorted versions of the original ones:

```
x1d=[ 1 -1  1 -1  1 -1 -1  1]
x2d=[ 1  1 -1 -1 -1  1 -1 -1]
x3d=[ 1  1  1 -1  1  1 -1  1]
```

`x1d` has a one bit error, `x2d` and `x3d` have two bit errors.

- Apply the update rule repeatedly until you reach a stable fixed point. Did all the patterns converge towards stored patterns?

You will probably find that `x1`, `x2` and `x3` are *attractors* in this network.

- How many attractors are there in this network? Hint: automate the searching.
- What happens when you make the starting pattern even more dissimilar to the stored ones (e.g. more than half is wrong)?

The number of iterations needed to reach an attractor scales roughly as $\log(N)$ with the network size, which means there are few steps for a network this small. If you want you can train a larger network to get slower convergence.

3.2 Sequential Update

So far we have only used a very small 8-neuron network. Now we will switch to a 1024-neuron network and picture patterns. Load the file *pict.dat*, which contains nine1024-dim patterns stored one after another. We can name them `p1`, `p2`, `p3`, `p4`, `p5`, `p6`, `p7`, `p8` and `p9`. To start with, learn the first three.

Since large patterns are hard to read as rows of numbers, please display these 1024-dim patterns as a 32×32 image.

- Can the network complete a degraded pattern? Try the pattern `p10`, which is a degraded version of `p1`, or `p11` which is a mixture of `p2` and `p3`.
- Clearly convergence is practically instantaneous. What happens if we select units randomly? Please calculate their new state and then repeat the process in the spirit of the original sequential Hopfield dynamics. Please demonstrate the image every hundredth iteration or so.

3.3 Energy

Can we be sure that the network converges, or will it cycle between different states forever?

For networks with a *symmetric connection matrix* it is possible to define an *energy function* or *Lyapunov function*, a finite-valued function of the state that always decreases as the states change. Since it has to have a minimum at least somewhere the dynamics must end up in an attractor[1]. A simple energy function with this property is:

$$E = -\sum_i \sum_j w_{ij} x_i x_j$$

- What is the energy at the different attractors?
- What is the energy at the points of the distorted patterns?
- Follow how the energy changes from iteration to iteration when you use the sequential update rule to approach an attractor.
- Generate a weight matrix by setting the weights to normally distributed random numbers, and try iterating an arbitrary starting state. What happens?
- Make the weight matrix symmetric (e.g. by setting `w=0.5*(w+w')`). What happens now? Why?

3.4 Distortion Resistance

How resistant are the patterns to noise or distortion? You can use the `flip(x,n)` function, which flips `n` units randomly. The first argument is a row vector, the second the number of flips.

```
>> p1dist = flip(p1,5);
>> vis(p1dist);
```

(You may have to do this several times to get an impression of what can happen)

Train a network with `p1, p2, p3`, add noise to a pattern, iterate it a number of times and check whether it has been successfully restored. Let the script run across 0 to 100% noise and plot the result. For speed, use the Little model rather than asynchronous updates.

- How much noise can be removed?

3.5 Capacity

Now add more and more memories to the network to see where the limit is. Start by adding `p4` into the weight matrix and check if moderately distorted patters can still be recognized. Then continue by adding others such as `p5`, `p6` and `p7` in some order and checking the performance after each addition.

[1] In the Little model it can actually end up alternating between two states with the same energy; in the Hopfield model with asynchronous updates the attractor will always be a single state.

- How many patterns could safely be stored? Was the drop in performance gradual or abrupt?
- Try to repeat this with learning a few random patterns instead of the pictures and see if you can store more.
- It has been shown that the capacity of a Hopfield network is around $0.138N$. How do you explain the difference between random patterns and the pictures?

Create 300 random patterns and train a 100-unit (or larger) network with them. After each new pattern has been added to the weight matrix, calculate how many of the earlier patterns remain stable (a single iteration does not cause them to change) and plot it.

- What happens with the number of stable patterns as more are learned?
- What happens if convergence to the pattern from a noisy version (a few flipped units) is used? What does the different behavior for large number of patterns mean?

The self-connections w_{ii} are always positive and quite strong; they always support units to remain at their current state. If you remove them and compare the curves from pure and noisy patterns for large number of patterns you will see that the difference goes away. In general it is a good idea to remove self-connections, even though it seems that this step lowers the memory performance. In fact, self-connections promote the formation of spurious patterns and have negative effect on noise removal capabilities.

- What is the maximum number of retrievable patterns for this network?
- What happens if you bias the patterns, e.g. use `sign(0.5+randn(300,100))` or something similar to make them contain more $+1$? How does this relate to the capacity results of the picture patterns?

3.6 Sparse Patterns

The reduction in capacity because of bias is troublesome, since real data usually is not evenly balanced.

Here we will use binary (0,1) patterns, since they are easier to use than bipolar (± 1) patterns in this case and it makes sense to view the "ground state" as zero and differing neurons as "active". If the average activity $\rho = (1/NP)\sum_\mu \sum_i x_i^\mu$ is known, the learning rule can be adjusted to deal with this imbalance:

$$w_{ij} = \sum_{\mu=1}^{P} (x_i^\mu - \rho)(x_j^\mu - \rho)$$

This produces weights that are still on average zero. When updating, please use the slightly updated rule

$$x_i \leftarrow 0.5 + 0.5 * \text{sign}(\sum_j w_{ij} x_j - \theta)$$

where θ is a bias term.

- Try generating sparse patterns with just 10% activity and see how many can be stored for different values of θ (use a script to check different values of the bias).
- What about even sparser patterns ($\rho = 0.05$ or 0.01)?

Good luck!