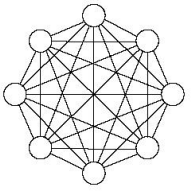
Neural Networks and Biological Modeling

Professor Wulfram Gerstner
Laboratory of Computational Neuroscience

Miniproject: Robustness of storage capacity in more realistic Hopfield networks

Assistants: Alex Seeholzer

Aim and motivations:



The Hopfield model is a simple, elegant and analytically tractable framework for the storage and associative retrieval of random, uncorrelated patterns. This elegancy stems partly from the symmetry of the weight matrix acquired from a Hebbian type correlation of pre- and postsynaptic firing activity. However, when compared to the connectivity of biological networks of neurons this symmetry seems quite unrealistic, since reciprocal connections of equal strength are very improbable. A second biologically implausible aspect of classical Hebbian weight matrices, where outgoing connection weights can have both positive and negative sign, is the violation of Dale's law – the biological principle that for each neuron, synaptic connections to postsynaptic neurons are either of the excitatory or inhibitory type, never both.

The aim of this miniproject is to investigate the robustness of the storage capacity of Hopfield networks, when the Hebbian weight matrices are altered to reflect these biological constraints. This will include the implementation of a Hopfield network in a programming language of your choice and the systematic evaluation of its properties. For an easy installation of a complete Python distribution, including the crucial scientific packages numpy and pylab/matplotlib, you can use the Enthought Python distribution[1]. For a quick recap of the essential features of Hopfield networks check the week 6 reading assignment available on Moodle.

Description: We consider here the classical Hopfield model at zero temperature, consisting of N fully connected binary nodes (in the following also called pixels) $S_i(t) \in \{-1; 1\}$ with no autaptic connections (i.e. $w_{ii} = 0$). P random and uncorrelated memory patterns $\xi^\mu \in \{-1; 1\}^N$ ($\xi_i^\mu = 1$ with probability $\frac{1}{2}$ and $\xi_i^\mu = -1$ otherwise) are stored in the network by the weight matrix given in the standard Hebbian form:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{P} \xi_i^\mu \xi_j^\mu . \tag{1}$$

In each unit time step, all nodes in the network are updated *sequentially*, from node 1 to node N, by the dynamical rule

it might be advantageous to upade the N pixels randomly. If we wish to simulate the brain there is no reason why we should assume that the pixels are updates in order.

$$S_i = \text{sign}\left(\sum_{j=1}^{N} w_{ij} S_j \right), \tag{2}$$

where we define sign(0) = 1. This is also called *sequential asynchronous* updating, since updated node values can affect the update of other nodes during the same unit time step[2].

To retrieve a pattern ξ^μ from the network, do the following:

1. At $t = 0$ initialize the network in a state close to the pattern ξ^μ – to get this initial state, randomly flip a given percentage c of the pattern's pixels.
2. Advance the network in time by Eq. 2 until the network's dynamics relax to a stable state. You will have to define a convergence criterion.

[1] http://www.enthought.com, look for EPDFree

[2] Unit time is thus effectively discretized in steps of $\frac{1}{N}$. In each of these steps one of the N nodes is updated and this update takes in to account the current values of all other nodes.

Exercise 1: Basic implementation of the Hopfield network

1.1 Getting started

Asynchronous updating of the nodes ensures that the dynamics of the Hopfield model with symmetric weights ($w_{ij} = w_{ji}$) always converge to a fixed point. An energy function in this case is given by

$$H(t) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} S_i(t) S_j(t).$$

Implement the Hopfield network as described above. Shortly comment on your implementation. The overlap of the network in state $S(t)$ with a given pattern ξ^μ is defined by $\frac{1}{N}\sum_{i=1}^{N} \xi_i^\mu S_i(t)$. Set $N = 200, P = 5, c = .2$. For the first two unit time steps of the retrieval of one pattern, plot the changing values of the energy function and the overlap of the network with the pattern as the state of each node is sequentially updated.

Hint: A function for efficient matrix multiplication in python is provided by the `dot` function in the `numpy` package.

1.2 Normalized pixel distance

Using the overlap defined in 1.1, derive an expression for the percentage of pixels in the network state S which differ from the pattern ξ^μ. This is the (normalized) pixel distance.

1.3 Pattern retrieval

We define the *retrieval error* of a pattern ξ^μ in a Hopfield network as the normalized pixel distance of the network state S to the pattern ξ^μ after convergence, when retrieving the pattern ξ^μ as described above.

Set $N = 200$ and $c = 0.1$. Plot the mean retrieval error of a randomly chosen pattern averaged over different network realizations as a function of the dictionary size P. Average over enough network realizations to give a smooth curve and give error bars for the estimation of the mean (you can assume the variation to be normally distributed). From your data points and the chosen confidence interval, roughly estimate a maximal P at which patterns can be retrieved from the network with a mean retrieval error of less than 2%?

Hint: A function for efficient matrix multiplication in python is provided by the `dot` function in the `numpy` package.

Exercise 2: Capacity estimation

We now define the *capacity* of a Hopfield network of size N as the number of patterns $P_{\mathrm{N,max}}$ that can be stored, such that the mean retrieval error *averaged over all stored patterns* (one retrieval attempt each) is at most 2%. This yields the maximal load $\alpha_{\mathrm{N,max}} = \frac{P_{\mathrm{N,max}}}{N}$.

Set c=0.1. Calculate $\alpha_{\mathrm{N,max}}$ for at least 10 network realizations and state the mean together with confidence intervals. Do this for $N = 100$, 250 and one other larger network size. Shortly interpret the resulting values and compare with results from literature.

Hint: To calculate $P_{\max}$, successively add patterns and calculate the mean retrieval error over.

Exercise 3: Random asymmetry

To investigate the robustness of pattern retrieval against asymmetry in the weight matrix, consider now networks where for each pairs of nodes (i, j), the directed connection from node i to node j is cut with probability p_{cut}. This will introduce asymmetry in the Hebbian weight matrix Eq. 1.

Set c=0.1 and $N = 200$. As in Ex. 2 calculate and plot the mean $\alpha_{\text{N,max}}$ for varying p_{cut} with error bars (at least 10 repetitions). At which p_{cut} does the maximal load drop below 50% of the value estimated in Ex. 2? State the confidence interval.

Note: Bare in mind that the convergence assertion of Question 1.1 does not necessarily hold for $p_{\text{cut}} > 0$.

Exercise 4: Dale's law

Set c=0.1, $N = 200$ and turn off random asymmetry ($p_{\text{cut}} = 0$). Set $E \in [0, 1]$ to be the percentage of excitatory nodes.

For a given E, randomly split the network into an excitatory and an inhibitory subpopulation (of sizes $E \cdot N$ and $(1 - E) \cdot N$ respectively). Now enforce Dale's law on the network connectivity by setting "disallowed" connection weights in the standard Hebbian weight matrix Eq. 1 to zero. What percentage of the directed connections do you expect to be cut for $E = \frac{1}{2}$?

As in Ex. 2 calculate and plot the mean $\alpha_{\text{N,max}}$ for varying E with error bars (at least 10 repetitions). Interpret the results and compare the value for $E = \frac{1}{2}$ to your result from Ex. 3.