Neuronal Dynamics: Python Exercises

Professor Wulfram Gerstner
Laboratory of Computational Neuroscience, EPF Lausanne

Hopfield Network

Exercise 1

To test the theoretical results of the preceding exercises you will now simulate an associative memory network. You will use a model in which neurons are pixels and take the values of -1 (*off*) or +1 (*on*). The network can store a certain number of pixel patterns, which is to be investigated in this exercise. During a retrieval phase, the network is started with some initial configuration and the network dynamics evolves towards the stored pattern (attractor) which is closest to the initial configuration. The dynamics is that of equation:

$$S_i(t+1) = sgn\left(\sum_j w_{ij} S_j(t)\right) \tag{1}$$

In the Hopfield model each neuron is connected to every other neuron (full connectivity). The connection matrix is

$$w_{ij} = \frac{1}{N}\sum_\mu p_i^\mu p_j^\mu$$

where N is the number of neurons, p_i^μ is the value of neuron i in pattern number μ and the sum runs over all patterns from $\mu = 1$ to $\mu = P$. This is a simple correlation based learning rule (Hebbian learning). Since it is not a iterative rule it is sometimes called one-shot learning. The learning rule works best if the patterns that are to be stored are random patterns with equal probability for on (+1) and off (-1). In a large networks (N to infinity) the number of random patterns that can be stored is approximately 0.14 times N.
This session will also introduce you to the concept of classes or object oriented code. You can find a tutorial at

```
http://docs.python.org/tutorial/classes.html
```

Download `hopfield.py` and the folder `alphabet/` from book's webpage. Open ipython and type

```
>> import hopfield
```

The module contains two classes

- `hopfield_network` the network itself which contains *methods* to create patterns and run a retrieval phase
- `alphabet` which creates pixel patterns out of the gif files contained in `alphabet/`

Create an instance of your `hopfield_network` class (here of size 4×4 for instance)

```
>> hn = hopfield.hopfield_network(4)
```

Now you can create and store patterns using

```
>> hn.make_pattern()
```

(default is one random pattern with half of its pixels *on*) and test whether it is stored with

```
>> hn.run()
```

which, by defaults, runs the dynamics for the first pattern with no pixel flipped.

1. In 4×4 network: What is the experimental maximum number of random patterns the network is able to memorize? Store more and more random patterns and test retrieval of some of them. The first few patterns should be stored perfectly, but then the performance gets worse. Does it correspond to the theoretical maximum number of random patterns the network is able to memorize?
2. Increase the network size to 10×10. Repeat question 1
3. Store a finite number of random patterns, e.g. 8. How many wrong pixels can the network tolerate in the initial state so that it still settles into the correct pattern?
4. Try to store characters as the relevant patterns. How good is the retrieval? What is the reason? (to store characters see the doc string of the method `make_pattern`)
5. *(Bonus)* Try one of the preceding points in bigger networks. Try adding a smooth transfer function g to the neurons.