Artificial neural networks - Exercise session 3

Recurrent neural networks

2015-2016

1 Hopfield Network

A Hopfield recurrent network has one layer of N neurons with `satlins` transfer functions, and is fully interconnected: each neuron is connected to every other neuron. After initialization of all neurons (the initial input), the network is let to evolve in a synchronous way: an output at time t becomes an input at time $t+1$. Thus to generate a series of outputs we have to provide only one initial input. In the course of this dynamical evolution the network should reach a stable state (an attractor state). This is a configuration of neuron values which is not changed after an update of the network. Networks of this kind are used as models of associative memory. After initialization the network should evolve to the closest attractor state.

Creation of a Hopfield network with N neurons:

```
net = newhop(T);
```

where `T` is a $N \times Q$ matrix containing Q vectors with components equal to ± 1. This command will create a recurrent Hopfield network with stable points being the vectors from `T`. For a 2-neuron network with 3 attractors states `[1 1]`, `[-1 -1]`, `[1 -1]`, `T` has the form[1]:

```
T = [1 1; -1 -1; 1 -1]';
```

We can simulate a Hopfield network in two modes:

- Single step iteration:

  ```
  [Y, Pf, Ai] = sim(net, 3, [ ], Ai);
  ```

  or briefly

  ```
  Y = sim(net, 3, [ ], Ai);
  ```

  The `sim` command will take the input vectors one by one, use them as input for the network, iterate the network a single timestep, and output the new vectors. If `Y == Ai` it means that the columns of `Ai` are attractors of the network `net`. The second argument of `sim` is the number of vectors in the input matrix (three in this example).

- Iteration for multiple times:
  If we want to iterate the network for multiple timesteps starting from inputs in a matrix `Ai` we can write a loop around above `sim` command where we use the output as the new input, e.g.

  ```
  for i=1:50 Ai = sim(net, 3, [ ], Ai); end
  ```

[1]The operator `'` transposes the matrix or vector. It is often more clear to write down a matrix row by row and then transpose it so the rows become columns. In above example the equivalent would be `[1 -1 1; 1 -1 -1]`.

Demos

You can run some demos as:

`playshow demohop1`	A two neuron Hopfield network
`playshow demohop2`	A Hopfield network with unstable equilibrium
`playshow demohop3`	A three neuron Hopfield network
`playshow demohop4`	Spurious stable points

Exercises

- Create a Hopfield network with attractors states `T = [1 1; -1 -1; 1 -1]'` and an arbitrary number of neurons. Start with various initial vectors and check the obtained attractors after a sufficient number of iterations. Do there exist other attractor states besides the ones that have been programmed into the network? How long does it typically take to reach the attractor states?

- Execute script `rep2`. Modify this script to start from some particular points (e.g. of high symmetry) or to generate other numbers of points. Are the attractors states always those stored in the network at creation?

- Do the same for a three neuron Hopfield network. This time use script `rep3`.

- The function `hopdigit` creates a Hopfield network which has as attractors the handwritten digits $0, \cdots, 9$. Then to test the ability of the network to correctly retrieve these patterns some noisy digits are given to the network. Is the Hopfield model always able to reconstruct the noisy digits? If not, why?

  You can call the function by typing:

  ```
  hopdigit(noise,numiter)
  ```

  where:

  `noise` represents the level of noise that will corrupt the digits and is a number ≥ 0

  `numiter` is the number of iterations the Hopfiled network (having as input the noisy digits) will run.

  Try to answer the above question by playing with these two parameters.

2 Elman Network

An Elman network is a two layer network with `tansig` neurons in a hidden layer, `purelin` neurons in an output layer and feedback from the hidden layer to the input. The delay in the feedback is one time step. Thanks to this feedback Elman networks can detect and generate time-varying patterns.

Creation of the network:

```
net = newelm(P,T,nh);
```

`P` is the vector of inputs, `T` represents the target, `nh` is the number of neurons in the hidden layer. One can also change the type of neurons in the hidden and output layers by changing the corresponding transfer functions. For instance the following command:

```
net = newelm(P,T,5,{'tansig', 'logsig'});
```

generates an Elman network with a number of input neurons based on P (e.g. `P = rand(1,30)`), 5 `tansig` neurons in the hidden layer and a number of `logsig` neurons in the output layer according to T. To simulate the network we can use the usual command:

```
Y = sim(net, P);
```

If we want the network to perform some particular task we can train it using the functions `train`. First we have to generate a target sequence. In our case we can take for instance:

```
T = 0.5*P;
```

and then train the net with inputs `P` and targets `T`:

```
net = train(net, P, T);
```

To change the number of training epochs (default is 100) one can execute the command:

```
net.trainParam.epochs = ne;
```

where `ne` is the new number of epochs. To estimate the results of learning we can visualize targets and results of the simulation of a trained network.

```
Y = sim(net,P);
plot(P,T,'rx',P,Y,'bo');
```

Demos

`appelm1`	Amplitude detection using Elman network

Exercises

Generate a simple time-series. Design an Elman network that is able to model it. Train the network with prepared examples. Try different time-series, number of training examples, learning times (number of epochs) and architectures.

Example:
We want the network to learn the behaviour of the following *Hammerstein system* with state-space representation:

$$\begin{cases} x(t+1) &= 0.6x(t) + \sin(u(t)) \\ y(t) &= x(t) \end{cases}$$

where $u(t)$ is a stochastic input, $y(t)$ is the output and $x(t)$ is the state of the system.

We can generate in matlab such a time-series in the following way:

```
u(1)=randn; %random number drawn from a standard gaussian distribution
x(1)=rand+sin(u(1));
y(1)=0.6*x(1);
for i=2:n
    u(i)=randn;
    x(i)=0.6*x(i-1)+sin(u(i));
    y(i)=x(i);
end
```

Then we create an Elman network by using this command:

```
net = newelm(X,T,n_neurons); %create network
```

In the script `elmants2` this network is trained and tested. The results are displayed at the end. You can use the script as example in order to do the proposed exercises.

Functions and commands

`newelm(P,T,nh)`	creates an Elman network; `P` contains the inputs, `T` contains the targets, `nh` is the number of neurons in the hidden layer.
`train(net, P, T)`	trains a network net with inputs `P` and targets `T` written in a cell array form, uses one of the batch learning algorithms
`tansig, purelin`	transfer functions
`rand`	generates a random number from [0,1] as components, drawn from a uniform distribution
`rands`	the same as `rand` but random numbers are taken from [-1,1]
`randn`	generates a random number drawn from a gaussian distribution with $\mu = 0$ and $\sigma = 1$.

3 Report

Write a report of maximum 2 pages (including text + figures) to discuss:

- the working principle of the Hopfield network (attractors, spurious states etc.) on the handwritten digits dataset. Use the function `hopdigit` as an example.
- the experimental results related to the application of the Elman network to the Hammerstein system data.

References

[1] H. Demuth and M. Beale, Neural Network Toolbox (user's guide), `http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/nnet.shtml`