Artificial neural networks - Exercise session 2

Recurrent neural networks

Sonny Achten, Bram De Cooman, Hannes De Meulemeester and David Winant

2022-2023

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), this is a configuration of neuron values which is not changed by 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.

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 `[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 = net([],[],Ai);
```

Multiple step iteration

```
Y = net({num_step},{},Ai);
```

with example inputs

```
Ai = [0.3 0.6; -0.1 0.8; -1 0.5]';
```

If `Y == Ai`, the columns of `Ai` are attractors of the network `net`.

Demos

Run the following demos (see Toledo):

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

[1]The operator `'` or T 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]`

Exercise

- Create a Hopfield network with attractors `T` $= [1\ 1; -1\ -1; 1\ -1]^{\mathrm{T}}$ and the corresponding number of neurons. You can use script `rep2` as a basis and modify it to start from some particular points (e.g. of high symmetry) or to generate other numbers of points. Start with various initial vectors and note down the obtained attractors after a sufficient number of iterations. Are the real attractors the same as those used to create the network? If not, why do we get these unwanted attractors? How many iterations does it typically take to reach the attractor? What can you say about the stability of the attractors?
- 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? What is the influence of the noise on the number of iterations?

  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 positive number.

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

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

2 Long Short-Term Memory Networks

2.1 Introduction: Time-series Prediction

A time series is a sequence of observations, ordered in time. Forecasting involves training a model on historical data and using them to predict future observations. A simple example is a linear auto-regressive model. The linear auto-regressive (AR) model of a time-series Z_t with $t = 1, 2, \ldots, \infty$ is given by:

$$\hat{z}_t = a_1 z_{t-1} + a_2 z_{t-2} + \cdots + a_p z_{t-p} \ , \tag{1}$$

with $a_i \in \mathbb{R}$ for $i = 1, \ldots, p$ and p the model lag. The prediction for a certain time t is equal to a weighted sum of the previous values up to a certain lag p. In a similar way, the nonlinear variant (NAR) is described as:

$$\hat{z}_t = f(z_{t-1}, z_{t-2}, \ldots, z_{t-p}) \ . \tag{2}$$

A depiction of these processes can be found in Figure 1. Remark that in this way, the time-series identification can be written as a classical black-box regression modeling problem:

$$\hat{y}_t = f(x_t) \ , \tag{3}$$

with $y_t = z_t$ and $x_t = [z_{t-1}, z_{t-2}, \ldots, z_{t-p}]$.

2.2 Neural network

The Santa Fe data set is obtained from a chaotic laser which can be described as a nonlinear dynamical system. Given are 1000 training data points. The aim is to predict the next 100 points (it is forbidden to include these points in the training set!). The training data are stored in lasertrain.dat and are shown in Figure 2a. The test data are contained in laserpred.dat and shown in Figure 2b.

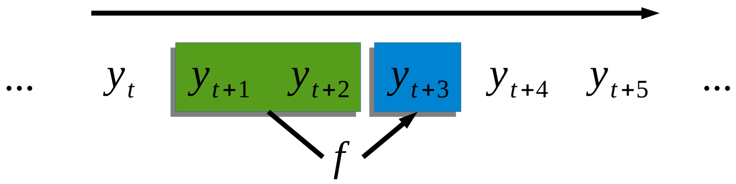
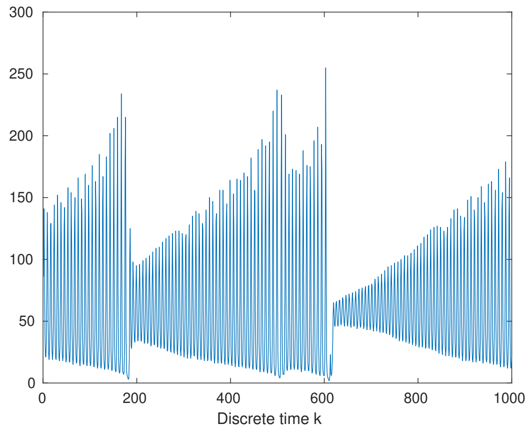
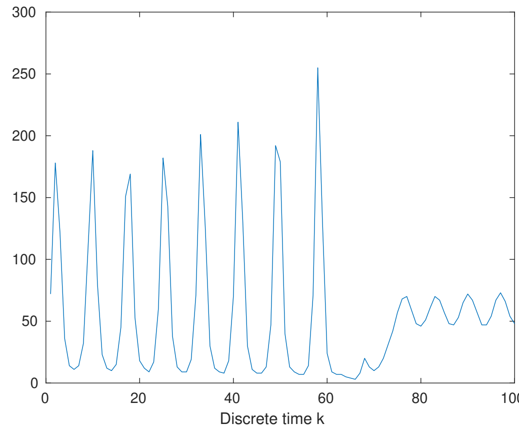


Figure 1: Schematic representation of the nonlinear auto-regressive model with lag $p = 2$.



(a) Training set



(b) Test set

Figure 2

Exercise

Train a MLP with one hidden layer **after standardizing** the data set. The training is done in feedforward mode:

$$\hat{y}_k = w^{\mathrm{T}} \tanh(V[y_{k-1}; y_{k-2}; ...; y_{k-p}] + \beta). \tag{4}$$

In order to make predictions, the trained network is used in an iterative way as a recurrent network:

$$\hat{y}_k = w^{\mathrm{T}} \tanh(V[\hat{y}_{k-1}; \hat{y}_{k-2}; ...; \hat{y}_{k-p}] + \beta). \tag{5}$$

To format the data you can use the provided function `getTimeSeriesTrainData`. Make sure you understand what the function does by trying it out on a small self-made toy example. To predict the test set you will have to write a `for` loop that includes the predicted value from the previous timestep in the input vector to predict the next timestep. Investigate the model performance with different lags and number of neurons. Explain clearly how do you tune the parameters and what is the influence on the final prediction. Which combination of parameters gives the best performance (RMSE) on the test set?

2.3 Long short-term memory network

Long Short Term Memory networks, usually just called "LSTMs", are a special kind of RNN, capable of learning long-term dependencies [2]. LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Those gates act on the signals they receive, and similar to the neural network's nodes, they block or pass on information based on its strength and importance, which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent.

Demo

Study the following example, where an LSTM is build to predict the monthly cases of chickenpox by running `openExample('nnet/TimeSeriesForecastingUsingDeepLearningExample')`.

Exercise

Based on the previous demo, try to model the Santa Fe data set.

- Train the LSTM model and explain the design process. Discuss how the model looks, the parameters that you tune, ... What is the effect of changing the lag value for the LSTM network?
- Afterwards try to predict the test set. Use the `predictAndUpdateState` function to predict time steps one at a time and update the network state at each prediction. For each prediction, use the previous prediction as input to the function.
- Compare results of the recurrent neural network with the LSTM. Which model do you prefer and why?

3 Report

Write a report of maximum 4 pages (including text and figures) to discuss the exercises in sections 1 and 2.

References

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

[2] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.