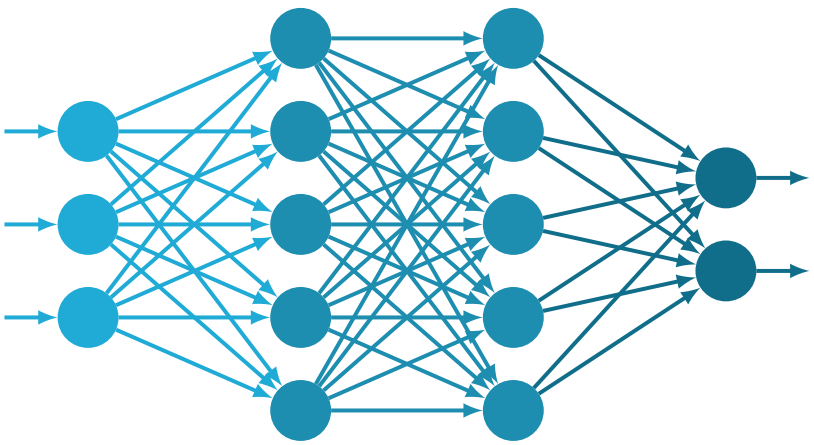
KU LEUVEN

Artificial Neural Networks & Deep Learning

Exercise Session 2

Prof. Dr. ir. Johan A. K. Suykens

Assistants
Sonny Achten
Bram De Cooman
David Winant
Xinjie Zeng



2023-2024

SESSION 2

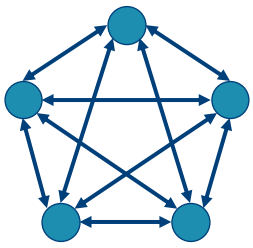
Recurrent Neural Networks

Note: There is no need to repeat the information provided during the guided session. Answer the questions as concise and accurate as possible. Show your understanding of the concepts and methods by linking your observations to the theory. Use graphical and/or tabular presentations where needed. The *page limit* for this exercise is *4 printed pages*, including figures.

This assignment focuses on recurrent neural networks and their applications as associative memory for pattern denoising or recognition, and as nonlinear autoregressive models for time-series prediction. For the former application, we study the Hopfield network, which is tested on a dataset of noisy handwritten digits. The latter application is demonstrated on a dataset of chaotic laser data, where we use multi-layer perceptrons and long short-term memory networks to predict the continuation of the time series.

2.1 HOPFIELD NETWORK

A Hopfield network [1, 2] has one layer of N neurons and is a fully interconnected recurrent neural network: each neuron is connected to every other neuron. After initialization of all neurons (the initial input), the network is let to evolve: 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), which is a configuration of neuron values that is not changed by subsequent updates of the network. Networks of this kind are used as models of associative memory. After initialization the network should evolve to the closest attractor.



To answer the questions in this section, you will need the notebook `hopfield.ipynb` that you can find on toledo, or alternatively using this colab link.

Exercises

1. Create a Hopfield network with target patterns $[1, 1]$, $[-1, -1]$ and $[1, -1]$ and the corresponding number of neurons. Simulate the state evolution for various input vectors (e.g. random points or points of high symmetry) 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?

2. Do the same for a three neuron Hopfield network, this time for the target patterns $[1, 1, 1]$, $[-1, -1, 1]$ and $[1, -1, -1]$.

3. Create a higher dimensional Hopfield network which has as attractors the handwritten digits from 0 to 9. Test the ability of the network to correctly retrieve these patterns when some noisy digits are given as input to the network. Try to answer the below questions by playing with these two parameters:

   - `noise_level` represents the level of noise that will corrupt the digits and is a positive number.
   - `num_iter` is the number of iterations the Hopfield network (having as input the noisy digits) will run.

   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?

2.2 TIMESERIES 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

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

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

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

A depiction of this process can be found in Figure 2.1. Remark that in this way, the time-series identification can be written as a classical black-box regression modeling problem $\hat{y}_t = f(x_t)$, with $y_t = z_t$ and $x_t = [z_{t-1}, z_{t-2}, \ldots, z_{t-p}]$. When preparing the dataset and applying train/validation/test splits, it is important to prevent *data leakage* by respecting the temporal information flow. More precisely, a datapoint z_t should not be part of two splits — either as input x_t or target y_t — and training (or validation) sets should not contain datapoints that occur after test datapoints.

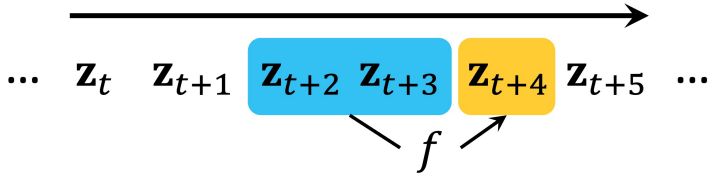


Figure 2.1: Schematic representation of the nonlinear auto-regressive model with lag $p = 2$.

To answer the questions in this section, you will need the notebook `time_series.ipynb` that you can find on toledo, or alternatively using this colab link.

Santa Fe Laser Dataset

The Santa Fe laser dataset is obtained from a chaotic laser which can be described as a nonlinear dynamical system. The first 1000 data points can be used for training and validation purposes. The aim is to predict the next 100 points (it is forbidden to include these points in the training or validation sets!). Both datasets are stored in `SantaFe.npz` and are shown in Figure 2.2. Before training any of the models, it is important to normalize the data, such that it has zero mean and unit variance.

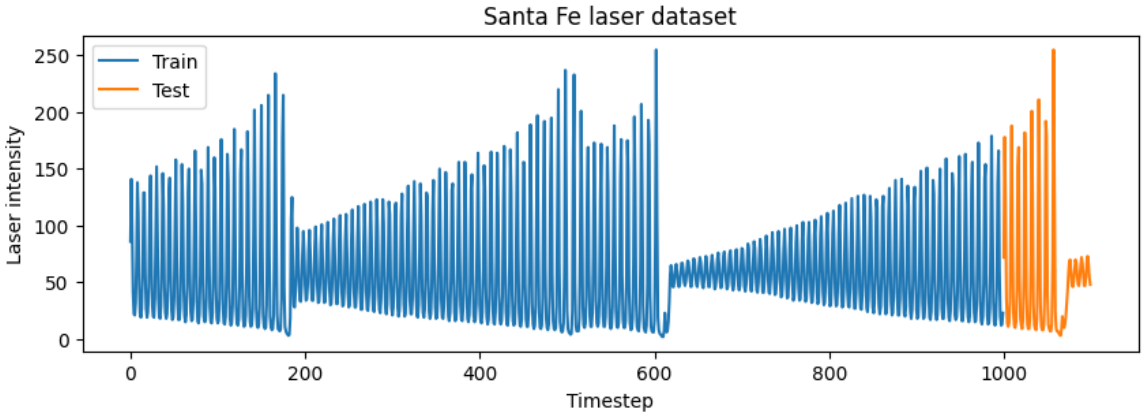


Figure 2.2: Visualization of the Santa Fe laser dataset.

Multilayer Perceptron

The first model we investigate is an MLP with one hidden layer. The training is done in feedforward mode:

$$\hat{z}_t = w^\top \tanh(V[z_{t-1}; z_{t-2}; ...; z_{t-p}] + \beta). \tag{2.1}$$

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

$$\hat{z}_t = w^\top \tanh(V[\hat{z}_{t-1}; \hat{z}_{t-2}; ...; \hat{z}_{t-p}] + \beta). \tag{2.2}$$

To format the timeseries for training, you can use the provided function `prepare_timeseries`. Make sure you understand what this function does by trying it out on a small toy example. To make predictions, a `for` loop needs to be implemented that includes the predicted value from the previous timestep in the input vector to predict the next timestep.

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 [3]. 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 the error, and adjusting weights via gradient descent. As this is a stateful model, make sure to reset the internal state using the `reset_states` function before making new predictions.

Exercises

1. Train an MLP with one hidden layer. Investigate the model performance with different lags and number of neurons. Discuss how the model looks and explain clearly how you tune the parameters and what the influence on the final prediction is. Which combination of parameters gives the best performance (MSE) on the test set?
2. Do the same for the LSTM model and explain the design process. What is the effect of changing the lag value for the LSTM network?
3. Compare the results of the recurrent MLP with the LSTM. Which model do you prefer and why?

REFERENCES

[1] J. J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554.

[2] J. J. Hopfield. “Neurons with graded response have collective computational properties like those of two-state neurons.” In: *Proceedings of the National Academy of Sciences* 81.10 (1984), pp. 3088–3092. DOI: 10.1073/pnas.81.10.3088.

[3] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.