# Recurrent Neural Networks

**James O'Reilly**
james.oreilly@student.kuleuven.be

## Introduction

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. In this assignment we will first discuss Hopfield networks, before exploring the use of MLPs and long short term memory networks (LSTMs) for time-series prediction.

## 1 Hopfield Networks

Hopfield networks serve as content-addressable memory systems with binary threshold nodes. They are often used as a model of associative memory. The network consists of a single layer of fully interconnected neurons. These neurons update synchronously, with the output of timestep $t$ as the input in timestep $t+1$. A single input therefore gives a series of output. After initialisation, the network will evolve dynamically toward a stable state (an attractor). One can choose the attractor states of the network, and then model the dynamic behaviour of the network with different initialisation states. Sometimes the network has attractors which are not explicit. Here we explore the convergence and attractors of simple Hopfield networks with two or three neurons.

### 1.1 A Two-Neuron Hopfield Network

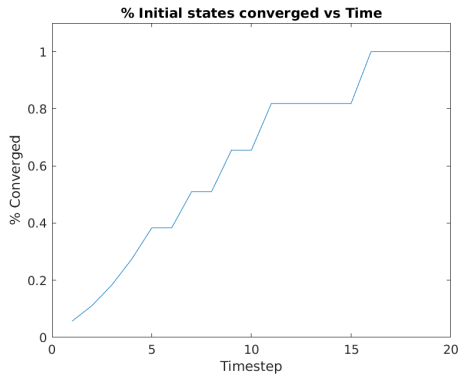
We create a two-neuron Hopfield network with three explicit attractors given by the matrix

$$T = \begin{bmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \end{bmatrix}$$

where each row is an attractor. The network was then initialised with other vectors to understand the dynamics of this network – including the presence of other attractor states that were not explicitly set, and also the number of timesteps taken to converge. To fully explore the space of initial vectors, a grid vectors $(x, y)$ was used, with $x, y \in [1, -1]$. The use this grid has benefits of random initial vectors for finding implicit attractor states as using highly-symmetric points allows us to see saddle points which are situated exactly halfway between the explicit attractors. Using this grid, another set $S$ of attractor states was found:

$$S = \begin{bmatrix} -1 & 0 \\ 1 & 0 \\ -1 & 1 \\ 0 & -1 \\ 0 & 0 \\ 0 & -1 \end{bmatrix}$$

At each timestep, the percentage of initial vectors which had reached one of the stable states given in $S$ was calculated. The results of this are shown in Figure 1.



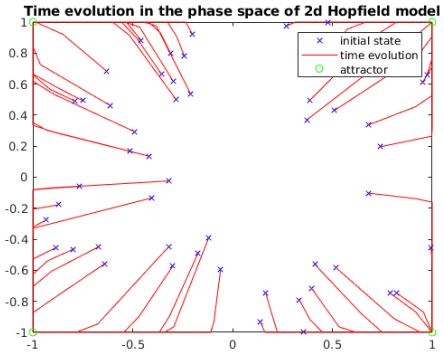



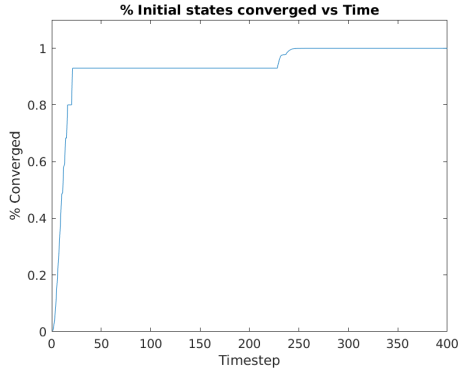

Figure 1: Convergence for the two neuron Hopfield network, along with the trajectories for random initial states.

## 1.2 A Three-Neuron Hopfield Network

Similarly, we create a three neuron Hopfield network with three explicit attractors given by the matrix

$$T = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & -1 & -1 \end{bmatrix}$$

Initialising the network with vectors from a three-dimensional grid, no implicit attractor states were found. However, looking at the graph for convergence over time (see Figure 2), it shows that there were a number of initial states for which it took over 200 iterations to converge to one of the explicit attractors. Also given are the trajectories of ten randomly generated initial states.



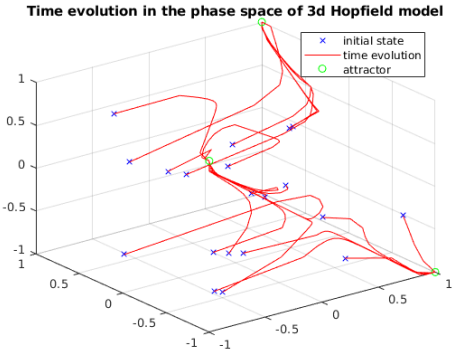




Figure 2: Convergence for the three neuron Hopfield network, along with the trajectories for ten random initial states.

## 1.3 A Hopfield Network for Handwritten Digits

We create a Hopfield network which has as attractors the handwritten digits $0, \ldots, 9$. Then to test the ability of the network to correctly retrieve these patterns some noisy digits are given to the network. As the noise is increased, the ability of the network to correct corrupted digits falls off quite rapidly, particularly in the case of numbers which are similar to others (both numbers have loops, a vertical line, etc.), and so the network is not always able to reconstruct corrupted digits. Increasing the number of iterations improved performance but even with more than 1000 iterations, some digits could not be reconstructed for high levels of noise. The network would likely perform better with higher resolution images (more pixels), as this effectively increases the 'distance' between the attractor states, which would mean more noise is necessary to move from one state to another. Figure 3 shows the attempted reconstruction of corrupted digits.

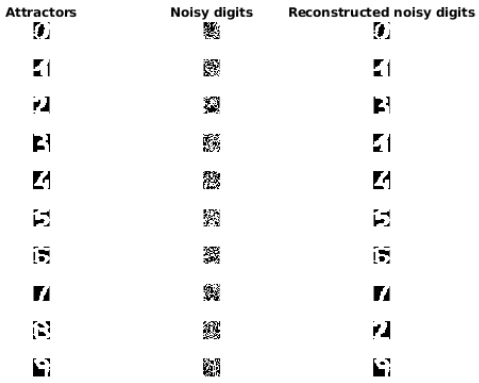



Figure 3: Attempted reconstruction of corrupted digits with a noise level of 6 and 10000 iterations.

# 2 Long Short-Term Memory Networks

In this section I give a brief introduction to time-series prediction, before trying make predictions about series data using a simple multi-layer perceptron with one hidden layer and then using an LSTM.

## 2.1 Time-Series Prediction using a Multi-layer Perceptron

The data being used is from a chaotic laser which can be described as a non-linear dynamical system. Given 1000 training data points, the aim is to predict the next 100 points. The training and test data are given in Figure 4.

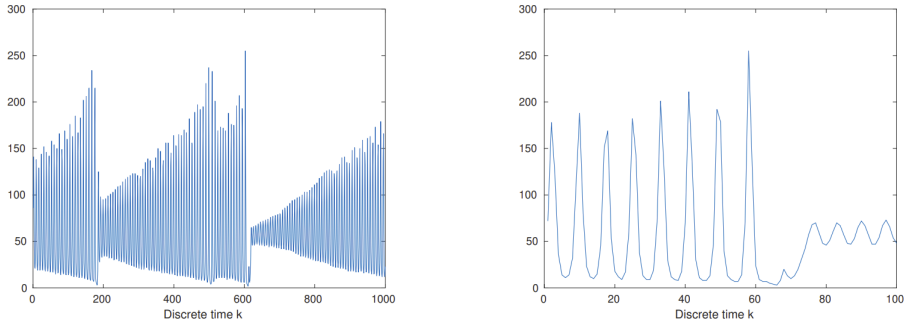



Figure 4: Training and test data from the chaotic laser.

First the data is standardised so that it has zero-mean and unit variance. After the dataset is standardised, an MLP with one hidden layer is trained. Training is done in feedforward mode:

$$\hat{y}_{k+1} = w^{\mathrm{T}} \tanh\left(V\left[y_k; y_{k-1}; \ldots; y_{k-p}\right] + \beta\right)$$

where $p$ is the lag – the number of previous timesteps considered. In order to make predictions, the trained network is used in an iterative way as a recurrent network:

$$\hat{y}_{k+1} = w^{\mathrm{T}} \tanh\left(V|\hat{y}_k; \hat{y}_{k-1}; \ldots; \hat{y}_{k-p}] + \beta\right)$$

A single-layer MLP with was trained on this data using Levenberg-Marquardt with Bayesian regularisation. The network was trained for 100 epochs with different lag values and different numbers of neurons in the hidden layer. While the performance of the network improved both as the number of neurons and the size of the lag was increased, the network originally could not capture the qualitative behaviour of the laser (see Figure 5). However, after increasing the lag to 30 and the number of hidden neurons to 50, the network was able to predict

the initial 'drop' after a series of increasing spikes (see Figure 5). The qualitative behaviour of the laser after the drop, however, was not predicted properly and behaved far too erratically. Perhaps this could be solved by either increasing the number of neurons, epochs or inputs, but I was reaching the limits of my laptop and didn't pursue this any further.

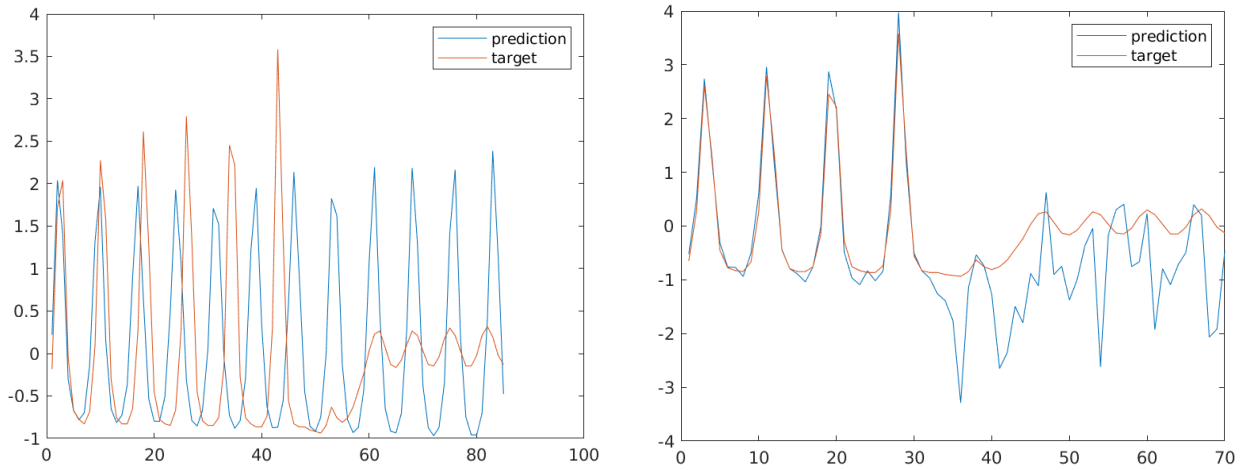



Figure 5: Target vs prediction plots. In the first plot, the network was trained for 100 epochs with a lag value of 15 and 30 hidden neurons. In the second plot, the network was trained for 100 epochs with a lag value of 30 and 50 hidden neurons.

## 2.2 Introduction to Long Short-Term Memory Networks

Long Short Term Memory networks, usually just called LSTMs, are a special kind of RNN, capable of learning long-term dependencies. 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 computers 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 networks 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, back-propagating error, and adjusting weights via gradient descent.

## 2.3 Training an LSTM on the Laser Data

The network architecture consists of a sequence input layer, an LSTM layer with 200 hidden units, a fully connected layer, and a regression layer. The network was trained with Adam for 500 epochs, and the learning rate is decreased after 250 epochs. To prevent gradients from exploding, a gradient threshold of 1 is used. The network was first trained with a lag of 1. That is, at each time step of the input sequence, the LSTM network learns to predict the value of the next time step. This network was then used to forecast the next 100 timesteps, without updating with observed values. The prediction and errors are shown in Figures 6a and 6c. Note that it is not able to accurately predict the qualitative behaviour of the laser. The network was then used to predict the next 100 timesteps, except the network state was updated after every prediction. The prediction and errors are shown in Figures 6b and 6d. By updating the states after every prediction, we are able to accurately predict the 'drop' in the signal given by the laser.

Ultimately I could not evaluate the effect of using different lag values for the LSTM as I had trouble formatting the data and training the LSTM with lag. My intuition would be that the RMSE for the prediction would be lower as the lag increased, and that in the case of forecasting future timesteps without updating the network state, it might have been possible to predict the qualitative behaviour correctly with sufficient lag.

Comparing the results from the LSTM to the recurrent neural network, it is clear that the LSTM performs better. Furthermore, the LSTM took less time to train – in order for the recurrent network to adequately capture the qualitative behaviour of the signal, it must be trained with a large number of inputs (large lag

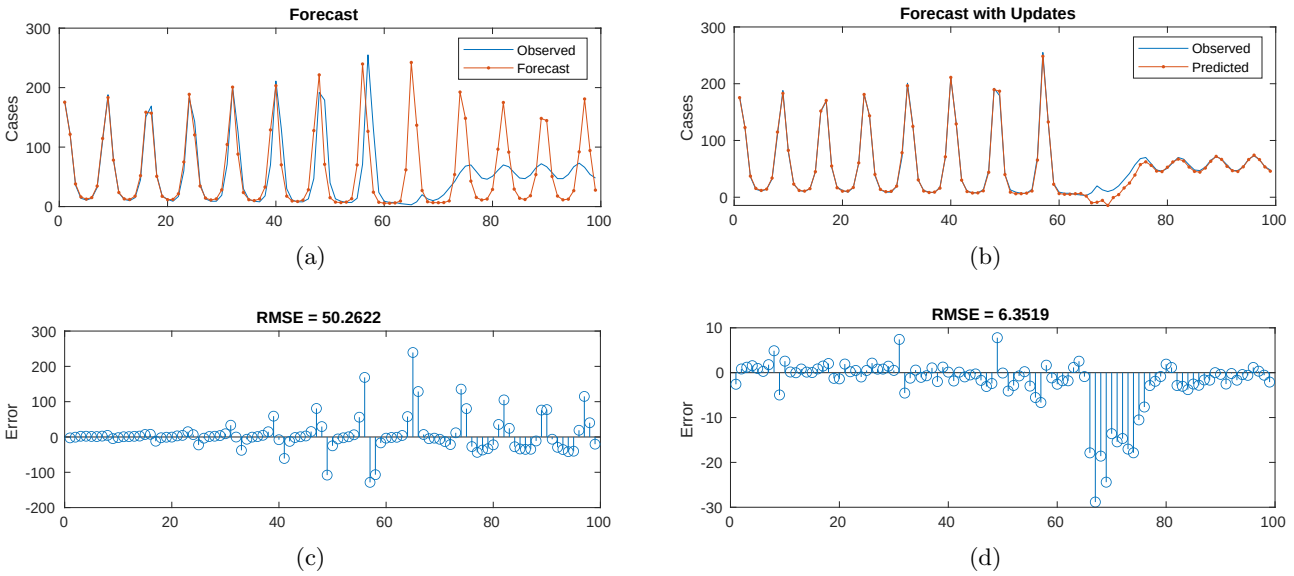



Figure 6: Predictions and RMSE for both the forecast *(left)* and updating the network with observed values *(right)*.

value). Whereas, in the case of the LSTM, we were able to accurately predict the signal using a lag value of one, and so the training time was much shorter.