Recurrent Neural Networks

Neural Computation : Lecture 12

1. Recurrent Neural Network Architectures
2. State Space Models and Dynamical Systems
3. Backpropagation Through Time
4. The NARX Model
5. Computational Power of Recurrent Networks
6. Applications – Reading Model, Associative Memory
7. Hopfield Networks and Boltzmann Machines

Recurrent Neural Network Architectures

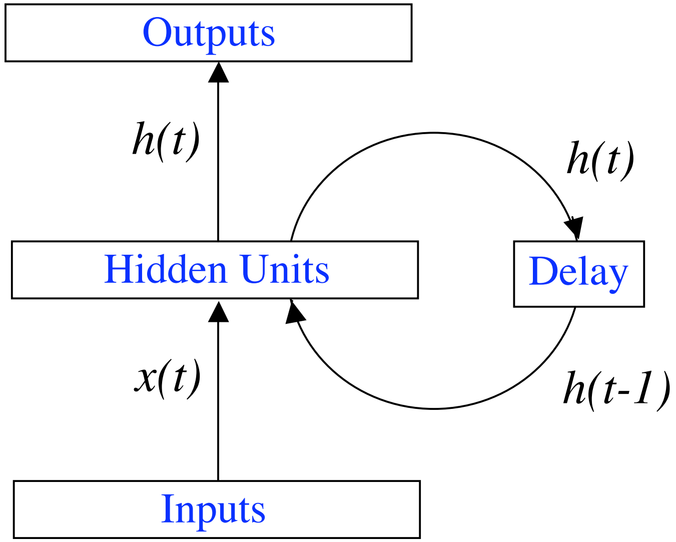
The fundamental feature of a ***Recurrent Neural Network (RNN)*** is that the network contains at least one ***feed-back connection***, so the activations can flow round in a loop. That enables the networks to do ***temporal processing*** and ***learn sequences***, e.g., perform sequence recognition/reproduction or temporal association/prediction.

Recurrent neural network architectures can have many different forms. One common type consists of a standard Multi-Layer Perceptron (MLP) plus added loops. These can exploit the powerful non-linear mapping capabilities of the MLP, and also have some form of ***memory***. Others have more uniform structures, potentially with every neuron connected to all the others, and may also have stochastic activation functions.

For simple architectures and deterministic activation functions, learning can be achieved using similar gradient descent procedures to those leading to the back-propagation algorithm for feed-forward networks. When the activations are stochastic, simulated annealing approaches may be more appropriate. The following will look at a few of the most important types and features of recurrent networks.

A Fully Recurrent Network

The simplest form of ***fully recurrent neural network*** is an MLP with the previous set of hidden unit activations feeding back into the network along with the inputs:



Note that the time *t* has to be ***discretized***, with the activations updated at each time step. The time scale might correspond to the operation of real neurons, or for artificial systems any time step size appropriate for the given problem can be used. A ***delay unit*** needs to be introduced to hold activations until they are processed at the next time step.

The State Space Model

If the neural network inputs and outputs are the vectors *x(t)* and *y(t)*, the three connection weight matrices are W_{IH}, W_{HH} and W_{HO}, and the hidden and output unit activation functions are f_H and f_O, the behaviour of the recurrent network can be described as a ***dynamical system*** by the pair of non-linear matrix equations:

$$h(t) = f_H\big(W_{IH}x(t) + W_{HH}h(t-1)\big)$$

$$y(t) = f_O\big(W_{HO}h(t)\big)$$

In general, the ***state*** of a dynamical system is a set of values that summarizes all the information about the past behaviour of the system that is necessary to provide a unique description of its future behaviour, apart from the effect of any external factors. In this case the state is defined by the set of hidden unit activations *h(t)*.

Thus, in addition to the input and output spaces, there is also a ***state space***. The ***order*** of the dynamical system is the dimensionality of the state space, the number of hidden units.

Stability, Controllability and Observability

Since one can think about recurrent networks in terms of their properties as dynamical systems, it is natural to ask about their *stability*, *controllability* and *observability*:

Stability concerns the boundedness over time of the network outputs, and the response of the network outputs to small changes (e.g., to the network inputs or weights).

Controllability is concerned with whether it is possible to control the dynamic behaviour. A recurrent neural network is said to be *controllable* if an initial state is steerable to any desired state within a finite number of time steps.

Observability is concerned with whether it is possible to observe the results of the control applied. A recurrent network is said to be *observable* if the state of the network can be determined from a finite set of input/output measurements.

A rigorous treatment of these issues is way beyond the scope of this module!

Learning and Universal Approximation

The ***Universal Approximation Theorem*** tells us that:

> Any non-linear dynamical system can be approximated to any accuracy by a recurrent neural network, with no restrictions on the compactness of the state space, provided that the network has enough sigmoidal hidden units.

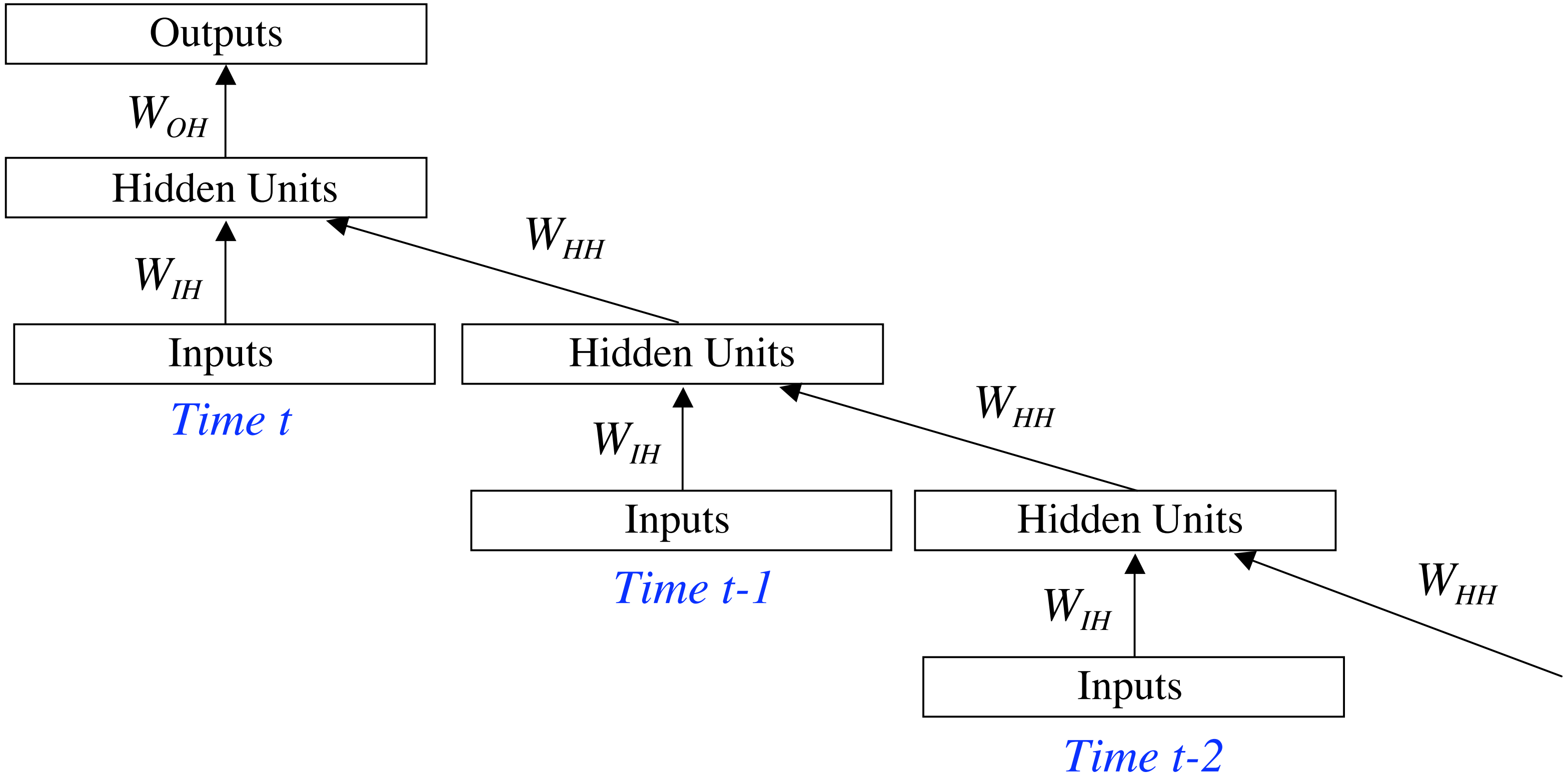
This underlies the computational power of recurrent neural networks.

However, knowing that a recurrent neural network can approximate any dynamical system does not tell us how to achieve it. As with feed-forward neural networks, we generally want them to learn from a set of training data to perform appropriately.

We can use ***Continuous Training***, for which the network state is never reset during training, or ***Epochwise Training***, which has the network state reset at each epoch. We now look at one particular learning approach that works for both training modes.

Unfolding Over Time

The recurrent network can be converted into a feed-forward network by ***unfolding over time***:



This means all the earlier theory about feed-forward network learning follows through.

Backpropagation Through Time

The Backpropagation Through Time ***(BPTT)*** learning algorithm is a natural extension of standard backpropagation that performs gradient descent on a complete unfolded network.

If a network training sequence starts at time t_0 and ends at time t_1, the total cost function is simply the sum over time of the standard error function $E_{sse/ce}(t)$ at each time-step:

$$E_{total}(t_0, t_1) = \sum_{t=t_0}^{t_1} E_{sse/ce}(t)$$

and the gradient descent weight updates have contributions from each time-step

$$\Delta w_{ij} = -\eta \frac{\partial E_{total}(t_0, t_1)}{\partial w_{ij}} = -\eta \sum_{t=t_0}^{t_1} \frac{\partial E_{sse/ce}(t)}{\partial w_{ij}}$$

The constituent partial derivatives $\partial E_{sse/ce}/\partial w_{ij}$ now have contributions from the multiple instances of each weight $w_{ij} \in \{W_{IH}, W_{HH}\}$ and depend on the inputs and hidden unit activations at previous time steps. The errors now have to be back-propagated through time as well as through the network.

Practical Considerations for BPTT

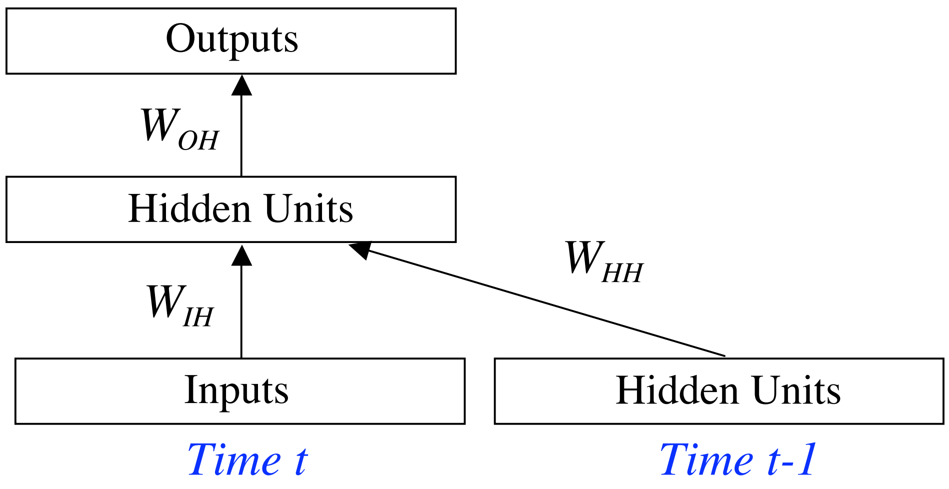
Even for the relatively simple recurrent network shown above, the unfolded network is quite complex, and keeping track of all the components at different points of time can become unwieldy. Most useful networks are even more problematic in this regard.

Typically the updates are made in an online fashion with the weights being updated at each time step. This requires storage of the history of the inputs and past network states at earlier times. For this to be computationally feasible, it requires ***truncation*** at a certain number of time steps, with the earlier information being ignored.

Assuming the network is stable, the contributions to the weight updates should become smaller the further back in time they come from. This is because they depend on higher powers of small feedback strengths (corresponding to the sigmoid derivatives multiplied by the feedback weights). This means that truncation is not as problematic as it sounds, though many times step may be needed in practice (e.g., ~30). Despite its complexity, BPTT has been shown many times to be an effective learning algorithm.

Simple Recurrent Networks

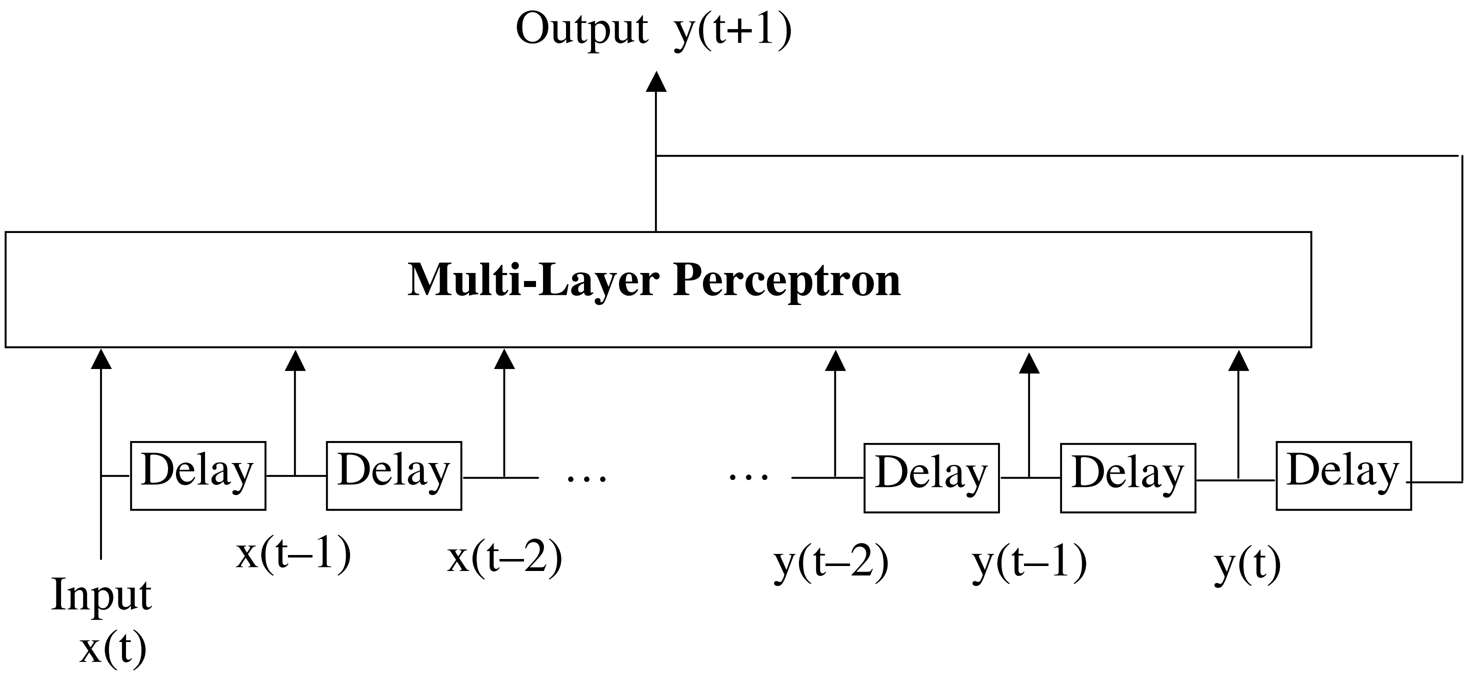
Truncating the unfolded network to just one time step reduces it to a so-called ***Simple Recurrent Network*** (which is also commonly known as an ***Elman network***):



In this case, each set of weights now only appears only once, so it is possible to apply the gradient descent approach using the standard backpropagation algorithm rather than full BPTT. This means that the error signal will not get propagated back very far, and it will be difficult for the network to learn how to use information from far back in time. In practice, this approximation proves to be too great for many practical applications.

The NARX Model

An alternative RNN formulation has a single input and a single output, with a delay line on the inputs, and the outputs fed back to the input by another delay line. This is known as the Non-linear Auto-Regressive with eXogeneous inputs (***NARX***) model:



It is particularly effective for ***Time Series Prediction***, where the target y(t+1) is x(t+1).

Computational Power of Recurrent Networks

Recurrent neural networks exemplified by the fully recurrent network and the NARX model have an inherent ability to simulate *finite state automata*. Automata represent abstractions of information processing devices such as computers. The computational power of a recurrent network is embodied in two main theorems:

Theorem 1

All Turing machines may be simulated by fully connected recurrent networks built of neurons with sigmoidal activation functions.

Proof: Siegelmann & Sontag, 1991, *Applied Mathematics Letters*, vol 4, pp 77-80.

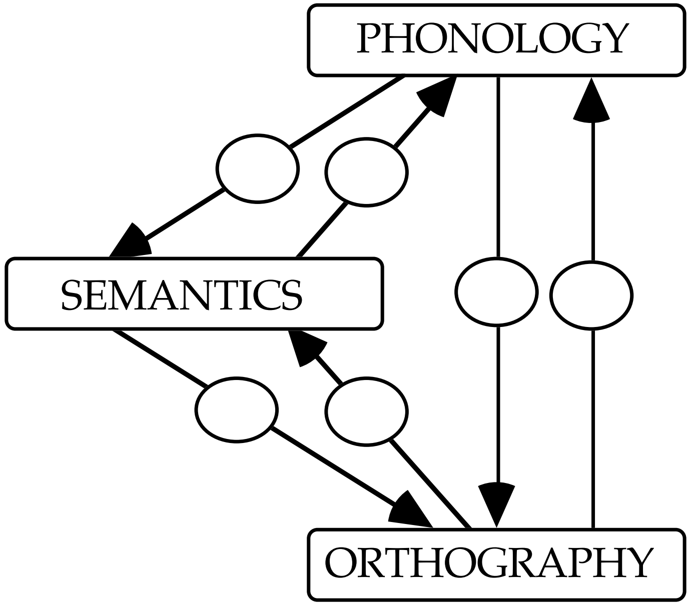
Theorem 2

NARX Networks with one layer of hidden neurons with bounded, one sided saturated (BOSS) activation functions and a linear output neuron can simulate fully connected recurrent networks with bounded one-sided saturated activation functions, except for a linear slowdown.

Proof: Siegelmann et al., 1997, *Systems, Man and Cybernetics, Part B*, vol 27, pp 208-215.

Application – Triangular Model of Reading

Any realistic ***model of reading*** must consider the interaction of orthography (letters), phonology (sounds) and semantics (meanings). A recurrent neural network of the form:



is required. The various sub-tasks (e.g., reading and spelling) need to be trained in line with human experience, and the dominant emergent pathways are found to depend on the nature of the language (e.g., English versus Chinese).

Application – Associative Memory

Associative Memory is the general idea of accessing memory through content rather than by address or location. One approach to this is for the memory content to be the pattern of activations on the nodes of a recurrent neural network. The idea is to start the network off with a pattern of activations that is a partial or noisy representation of the required memory content, and have the network settle down to the required content.

The basic associative memory problem can be stated as:

> Store a set of P binary valued patterns $\{\mathbf{t}^p\} = \{t_i^p\}$ in such a way that, when it is presented with a new pattern $\mathbf{s} = \{s_i\}$, the system (e.g., recurrent neural network) responds by producing whichever stored pattern $\mathbf{t}^p$ most closely resembles $\mathbf{s}$.

One could have separate input and output nodes to achieve this, or one could have a single set of nodes that act both as inputs and outputs. Either way, one needs to specify suitable node activation functions, and weight definition or update equations, so that the recurrent connections allow the learned memories to be accessed.

Hopfield Networks

The ***Hopfield Network*** ***or*** ***Hopfield Model*** is one good way to implement an associative memory. It is simply a fully connected recurrent network of N McCulloch-Pitts neurons. Activations are normally ±1, rather than 0 and 1, so the neuron activation equation is:

$$x_i = \mathrm{sgn}\left(\sum_j w_{ij} x_j - \theta_i \right) \qquad \text{where} \qquad \mathrm{sgn}(x) = \begin{cases} +1 & \text{if } \mathrm{x} \geq 0 \\ -1 & \text{if } \mathrm{x} < 0 \end{cases}$$

Unlike the earlier feed-forward McCulloch-Pitts networks, the activations here depend on time, because the activations keep changing till they have all settled down to some stable pattern. Those activations can be updated either ***synchronously*** or ***asynchronously***.

It can be shown that the required associative memory can be achieved by simply setting the weights w_{ij} and thresholds θ_j in relation to the target outputs $\mathbf{t}^p$ as follows:

$$w_{ij} = \frac{1}{N} \sum_{p=1}^{P} t_i^p t_j^p \quad , \qquad \theta_i = 0$$

A stored pattern $\mathbf{t}^q$ will then be stable if the neuron activations are not changing, i.e.

$$t_i^q = \operatorname{sgn}\left(\sum_j w_{ij} t_j^q - \theta_i\right) = \operatorname{sgn}\left(\sum_j \frac{1}{N} \sum_p t_i^p t_j^p t_j^q\right)$$

which is best analyzed by separating out the q term from the p sum to give

$$t_i^q = \operatorname{sgn}\left(t_i^q + \frac{1}{N} \sum_j \sum_{p \neq q} t_i^p t_j^p t_j^q\right)$$

If the second term in this is zero, it is clear that pattern number q is stable. It will also be stable if the second term is non-zero but has magnitude less than 1, because that will not be enough to move the argument over the step of the sign function sgn(x). In practice, this happens in most cases of interest as long as the number of stored patterns P is *small enough*. Moreover, not only will the stored patterns be stable, but they will also be ***attractors*** of patterns close by. Estimates of what constitutes a small enough number P leads to the idea of the ***storage capacity*** of a Hopfield network. A full discussion of Hopfield Networks can be found in most introductory books on neural networks.

Boltzmann Machines

A ***Boltzmann Machine*** is a variant of the Hopfield Network composed of N units with activations $\{x_i\}$. The state of each unit i is updated asynchronously according to the rule:

$$x_i = \begin{cases} +1 & \text{with probability } p_i \\ -1 & \text{with probability } 1 - p_i \end{cases}$$

where

$$p_i = \frac{1}{1 + \exp\left(-(\sum_{j=1}^{N} w_{ij} x_j - \theta_j) / T\right)}$$

with positive temperature constant T, network weights w_{ij} and thresholds θ_j.

The fundamental difference between the Boltzmann Machine and a standard Hopfield Network is the ***stochastic activation*** of the units. If T is very small, the dynamics of the Boltzmann Machine approximates the dynamics of the discrete Hopfield Network, but when T is large, the network visits the whole state space. Another difference is that the nodes of a Boltzmann Machine are split between visible input and output nodes, and hidden nodes, and the aim is to have the machine learn input-output mappings.

Training proceeds by updating the weights using the ***Boltzmann learning algorithm***

$$\Delta w_{ij} = -\frac{\eta}{T}\left(\left\langle x_i x_j \right\rangle_{fixed} - \left\langle x_i x_j \right\rangle_{free}\right)$$

where $\left\langle x_i x_j \right\rangle_{fixed}$ is the expected/average product of x_i and x_j during training with the input and output nodes fixed at a training pattern and the hidden nodes free to update, and $\left\langle x_i x_j \right\rangle_{free}$ is the corresponding quantity when the output nodes are also free.

For both Hopfield Networks and Boltzmann Machines one can define an ***energy function***

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} x_i x_j + \sum_{i=1}^{N} \theta_i x_i$$

and the network activation updates cause the network to settle into a local minimum of this energy. This implies that the stored patterns will be local minima of the energy. If a Boltzmann Machine starts off with a high temperature and is gradually cooled (known as ***simulated annealing***), it will tend to stay longer in the basins of attraction of the deepest minima, and have a good change of ending up in a global minimum at the end. Further details about Boltzmann Machines can be found in most introductory textbooks.

Restricted Boltzmann Machines and Deep Learning

One particularly important variation of the standard Boltzmann Machine is the ***Restricted Boltzmann Machine*** that has distinct sets of visible and hidden nodes, with no visible-to-visible or hidden-to-hidden connections allowed. This restriction makes it possible to formulate more efficient training algorithms, in particular, the ***Contrastive Divergence*** learning algorithm that performs an approximation to gradient descent.

These networks have proved effective in the field of ***Deep Learning***, and were what started much of the current interest in all forms of deep learning neural networks. Each layer of a many-layered feed-forward neural network can be efficiently pre-trained as an unsupervised Restricted Boltzmann Machine, and the whole network can then be fine-tuned using standard supervised Backpropagation.

Such use of semi-supervised feature learning in deep hierarchical neural networks is replacing hand-crafted feature creation in many areas. This is a major ongoing research area, with many new developments published each year.

Overview and Reading

1. We began by noting the important features of recurrent neural networks and their properties as fully fledged dynamical systems.
2. Then we studied a basic Fully Recurrent Network and unfolding it over time to give the Backpropagation Through Time learning algorithm.
3. This was followed by a brief overview of the NARX model and two theorems about the computational power of recurrent networks.
4. We looked at a Reading Model and Associative Memory as examples.
5. Finally, Hopfield Networks and Boltzmann Machines were introduced.

Reading

1. Haykin-2009: Sections 11.7, 13.7, 15.1 to 15.14
2. Hertz, Krogh & Palmer: Sections 2.1, 2.2, 7.1, 7.2, 7.3
3. Ham & Kostanic: Sections 5.1 to 5.9