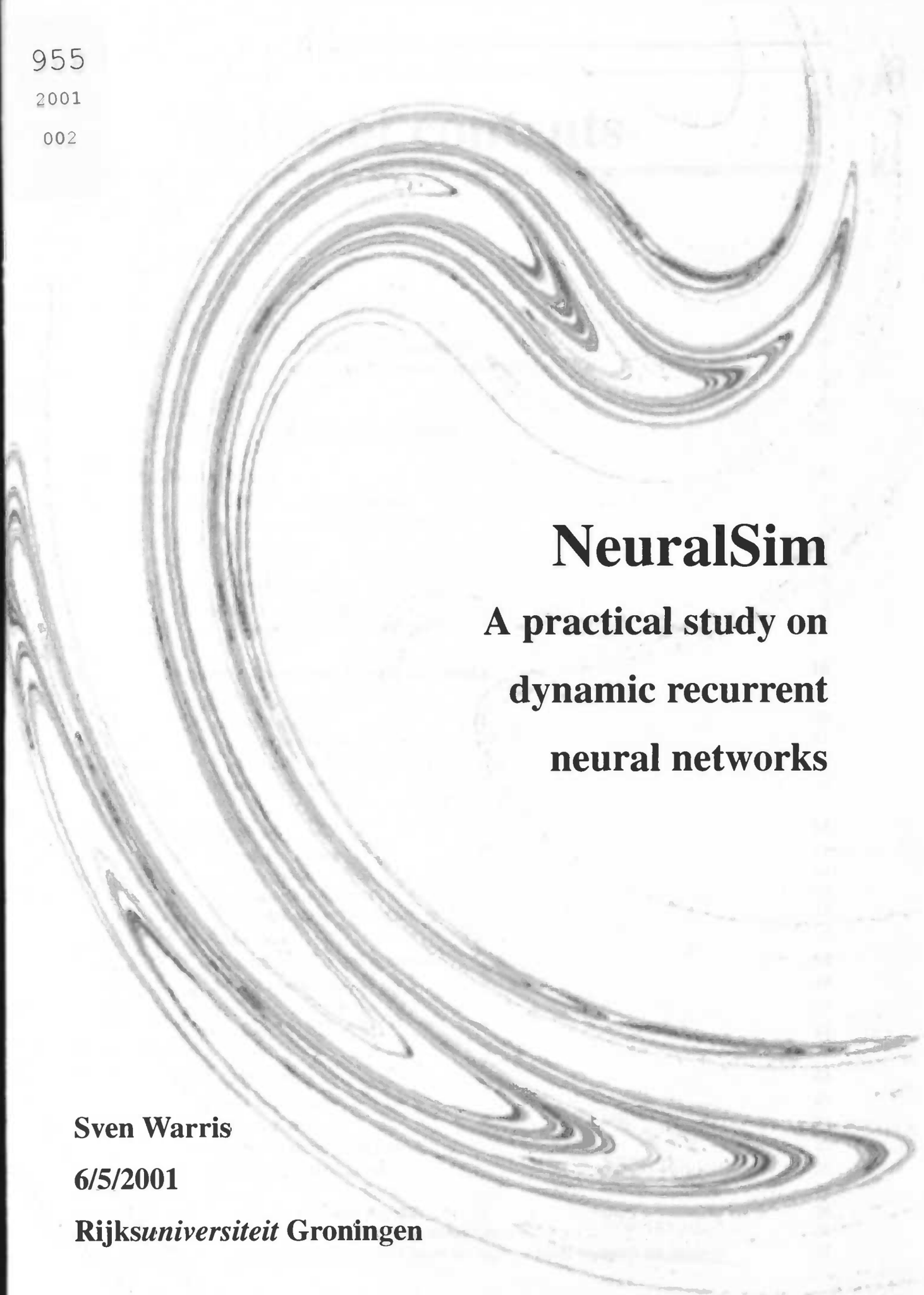


NeuralSim

A practical study on dynamic recurrent neural networks

Sven Warris

6/5/2001

Rijks*universiteit* Groningen

Table of contents

1 Aim

This chapter will give an overview of the aim of this document. It contains a short description of the type of neural network used. At the end of this chapter the experiments conducted with the software and their (scientific) significance will be outlined.

1.1 Setup of the project

The subsection NeuroBioPhysics of the Science Department at the Rijksuniversiteit Groningen conducts research in parts of the (insect) brain, mainly directed towards visual perception. However, there are no neural network simulators available which can be used to simulate the working of these particular neurons. The theoretical description of dynamic recurrent neural networks (DRNN) indicates that this type of neural network might be able to mimic the studied neurons.

A software implementation was needed to verify these claims and to see if a DRNN has any practical use. This document contains a description of the resulting software and the experiments that were conducted using this piece of software.

1.2 A dynamic recurrent neural network

The type of network used is called a dynamic recurrent neural network. Without going into much detail (these will follow later on), the terms are loosely described here:

- A neural network is a network of computational elements. These elements are more or less based on neural mechanisms which occur in the brain (hence the name neurons)
- If some or all of the neurons in a network not only have forward but also backward connections, the network is called recurrent
- A dynamic recurrent neural network is described using coupled differential equations over time

1.3 Experiments

To see if the program is correct it should be able to reproduce the experiments done by other researchers. In this light the first two experiments were set up:

1. Circle: can a network of 6 neurons (no inputs) create a circle
2. Figure-eight: can a network of 12 neurons (no inputs) learn to create a figure eight

These two experiments are well documented hence they are very suitable as test cases. The network built with the new piece of software should require a similar amount of learning epochs and have about the same performance. The results should be consistent to 'prove' the software is correct.

1.3.1 Effects of inhibition and excitation

A particular neuron in the brain can use only one type of neurotransmitter. Depending on the type of neurotransmitter a neuron will be an inhibitory or excitatory neuron. How does this affect the capabilities of the network? Will it be more flexible, hence learn faster/better or will it put constraints on the network? These questions will be evaluated using the experiments of the circle and the figure eight.

1.3.2 Effects of noise on the learnability of a circle

In practice the input and output signals are rarely without noise. It is therefore of much interest to see what effect white noise has on the overall learnability.

1.3.3 XOR

Although it is not the type of problem for which a dynamic recurrent network is most suited, the XOR has had a major influence on the development of neural networks in general. Therefore it is only natural that an experiment is run to see if a DRNN can solve this problem and if so, in what fashion.

1.3.4 Scalability

The software implementation needs to be tested to see if it is scalable and if so, to see how the learning time increases when more neurons are added to the network or when the number of data points in the input and target signals are increased.

1.4 The use of DRNN in pitch estimations

Speech recognition research is a type of research performed by many researchers around the world. Usually efforts are directed towards statistical training of some sort, for example training hidden Markov models. Although these training methods have no real biological foundation, they are currently the most promising techniques available. Neural networks are used in speech recognition, but the results vary.

This project will take a different aim at speech recognition. First, the pre-processing of the data files will not be done by standard fourier spectra analysis, but by a biologically plausible model of the animal cochlea. Second, it is not the aim of this project to start directly with speech recognition, but it will try to solve a small but important part of the speech recognition process: pitch estimation. Although it would be nice to get a network which can be used in practise, the main goal will be to see if it is feasable and if using a DRNN is a step forward in neural network speech recognition.

2 Introduction

The aim of this chapter is to give an informal description of neural networks and the generic recurrent neural network in particular. It is not the intention of the writer to give a full and detailed insights in the enormous amount of types of neural networks available.

2.1 Introduction to neural networks

This document is set up for the reader familiar with the concepts of neural networks. Although a short reminder will follow in the next few paragraphs, it is too short and too global to get a good grasp of the concepts. There are many good books available, so it would be advisable to read one of these first[3][8].

The inspiration for artificial neural networks comes from the brain. The brain is a highly complex, nonlinear and parallel information-processing system. This in contrast with modern day computers: they are linear and sequential. Because of these features even the brain of a small animal like the bat can perform control and recognition tasks at a speed unmatched by computers. The aim of artificial neural networks is to develop models of the neurons in the brain to enable computers to perform complex tasks on the one hand and to get insights in the working of the brain on the other.

An informal definition of a neural network viewed as an adaptive machine is as follows[3]:

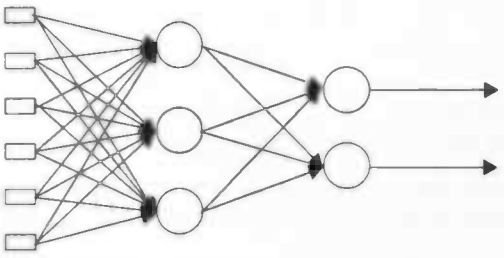
Definition 2.1

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. *Knowledge is acquired by the network through a learning process*
2. *Interneuron connection strengths known as synaptic weights are used to store the knowledge*

A neural network consists of several neurons connected to each other. These connections have synaptic weights, indicating the importance of the information flow from one neuron to the next. The knowledge of a network is therefore stored in these weights. Learning, or: adding knowledge, is the modification of the synaptic weights by minimizing a particular error or energy function.

Figure 2.1



A fully connected feedforward network with one hidden layer and output layer.

2.2 Common types of neural networks

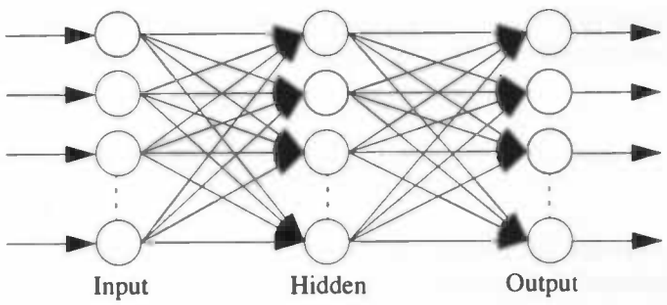
Neural computing has a long history. This history is not only very interesting, it also helps to get a better understanding of how and why the different types of networks were developed. Some networks are a response to other networks with a disadvantage while others were developed because a problem required a different type of neural network. In the next few paragraphs the historical development of the dynamic recurrent neural network (DRNN) is sketched.

2.2.1 Perceptron

The perceptron[8] is a simple artificial neural network. At first these networks were intended to be computational models of the retina. But following the development of new learning rules perceptrons came in use for all sorts of pattern recognition problems and other simular types of research.

A perceptron usually uses a binary output function (zero or one) or a bipolar output function (minus one or plus one).

Figure 2.2



A multilayer feedforward perceptron

For the binary version a single perceptron is described in the following mathematical equations. The output y_j of the perceptron is calculated by using the net input to the neuron as argument for the activation function, thus (with ω_{ij} the weight for input i connected to neuron j):

Eq 2.1

$$y_j = f(x_j)$$

$$x_j = \sum_{i=1}^{n} y_i \omega_{ij}$$

The weight update function is defined as:

Eq 2.2

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \Delta\omega$$

With $\Delta\omega$ as the change made to the weight and given as:

Eq 2.3

$$\Delta\omega_{ij} = \alpha(t_j - y_j)x_j$$

In Eq 2.3 the α is the learning rate with a range of $\langle 0, 1 \rangle$. t_j gives the target value for neuron j, hence $t_j - y_j$ gives the difference between the desired output and the real output of the neuron.

The major disadvantage of the perceptron is its inability to learn nonlinear classification problems. A function like AND or OR can be learned but the XOR for example cannot.

The perceptron learning algorithm is a steepest descent method: while learning it 'walks' through an energy landscape[1] looking for the deepest valley (a global minimum). But if it gets into a valley it cannot get out, while the solution presented by the network might not be the best solution

1. The energy landscape is defined by the error or energy function.

(hence a local minimum) at all: there could be another valley deeper than this one. The biggest problem is that you will probably never know if there is.

2.2.2 ADALINE

In 1959 Widrow developed the ADAptive LInear NEuron (ADALINE). An adaline unit has a bipolar output of -1 or +1. The inputs have no restriction: they may be binary or real valued. The transfer function gives +1 if the sum of the weighted inputs is greater than or equal to zero and a -1 if the input is less than zero. The adaptive learning algorithm is defined in Eq 2.4 and Eq 2.5.

Eq 2.4

$$E_{tot} = \sum_{p=1}^{P} E^p = \sum_{p=1}^{P} \sum_{j=1}^{m} (t_j^p - y_j^p)^2$$

Eq 2.4 gives the amount of energy (or error) of the network, with P as the number of patterns in the training set and m the amount of output cells. Now the change in weights can be calculated with η as a positive learning rate (the Widrow-learning rule):

Eq 2.5

$$\Delta\omega_{ij} = -\eta \frac{\partial E_{tot}}{\partial \omega_{ij}} = -\eta \sum_{p=1}^{P} \frac{\partial E^p}{\partial \omega_{ij}}$$

With $m = 1$ and by omitting the pattern subscript p Eq 2.5 can be rewriten by taking the partial derivatives of each term in Eq 2.5. This gives the Delta Learning rule for a single ADALINE cell:

Eq 2.6

$$\Delta\omega_i = \eta(t - y)x_i = \eta\varepsilon x_i$$

With ε as the error term $(t - y)$.

2.2.3 Backpropagation

In the previous section the delta rule is sketched. With some small adaptation it can also be applied to multi-layer networks (MADALINE). By introducing a differentiable function as the transfer function, a new method of learning can be developed based on the delta rule. In this new type of network the error is propagated back through the network from the output neurons to the input neurons. Between these two layers of neurons there can be any number of hidden layers. The principle remains that the weight adjustments start at the end of the network and finish at the input side of the network. This process stops when the error reaches to a minimum. The mathematics are straightforward and can be found in any book on neural networks[3][8]. The backpropagation equations can be summarized as follows:

Eq 2.7

$$\Delta\omega_{ji} = \eta \sum_{p=1}^{P} \delta_{out} H_{in}$$

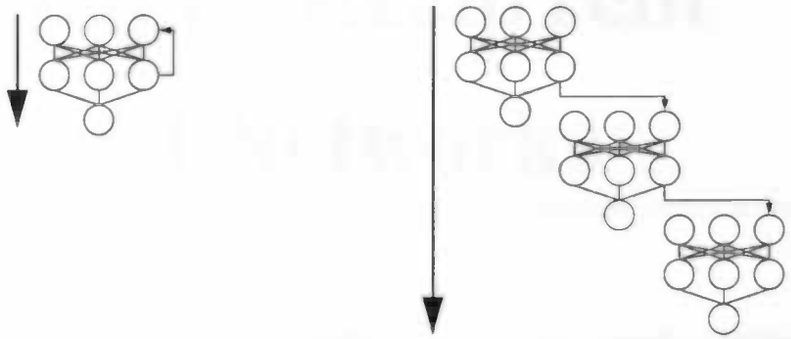
The form of δ varies with the type of layer for which it applies and H_{in} gives the current weighted input of the neuron.

2.3 Recurrent neural networks

A recurrent neural network is a neural network with not only feedforward connections but also feedback connections. Such a connection can be either from a layer back to its input layer or from a neuron back to itself. Recurrent networks based on the backpropagation algorithm can store a limited amount of temporal information.

A popular method of calculating recurrent neural networks is that of unfolding in time. At each timestep the network is enlarged with a copy of the current network. However, with a large amount of temporal input, the network grows out of bounds hence making this process inadequate for complex problems. For simple tasks it may proof usefull, because this way no new learning algorithm has to be designed.

Figure 2.3



A recurrent neural network (top-left) and the unfolding in time principle (right)[7]

It is also possible to redefine the output calculations with differential equations. The network now demonstrates dynamical behaviour in contrast to other networks which only demonstrate non-linear behaviour.

2.3.1 Dynamic recurrent networks

Dynamic recurrent neural networks use a different kind of output calculation than for example backpropagation networks. A DRNN is governed by a set of coupled differential equations:

Eq 2.8

$$T_i \frac{dy_i}{dt} = -y_i + F(x_i) + I_i$$

where y_i is the activation level of neuron i, $F(x)$ is the activation function (usually a squashing function) and x_i is given by:

Eq 2.9

$$x_i = \sum_{j=0}^{n} \omega_{ji} y_j$$

Chapter 3 gives a full description of DRNN's based on these equations.

3 Dynamic Recurrent Neural Networks

In this chapter the working of the dynamic recurrent neural network is explained in detail. Much of it comes from Draye[1] although not all the mathematical proof is given here. For proof, explanations and examples see Chapter 6 (by Mastebroek).

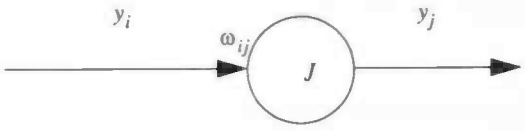
3.1 Basic neuron processes

A neuron in a DRNN receives information from an arbitrary number of inputs. These inputs can be divided into two different groups:

1. Inputs from neurons, including from itself
2. External inputs, with the bias as a special kind of external input

The output of a neuron is based upon its current weighted inputs:

Figure 3.1



The input flows from the left to the right. y_i is the output of neuron i *and y_j is the output of this neuron* (j). ω_{ij} *indicates the weight between neuron* i *and* j.

The neuron's output is governed by the following equations[1]:

Eq 3.1

$$T_i \frac{dy_i}{dt} = -y_i + F(x_i) + I_i$$

Where:

- y_i is the activation level of neuron i
- $F(\alpha)$ is the activation function (usually a squashing function)
- $x_i = \sum_{j=0}^{n} \omega_{ji} y_j$, the ω's are the weights of the n incoming signals for neuron i
- T_i is the time constant
- I_i is the external input of the neuron. If the input is a bias input, its value is constant (and usually set to 1)

3.2 Learning function

If a network is to learn some temporal behaviour an error function is needed. This function is used

1. This is called the propagation or motion equation of the network

in calculating the new weights and time constants of every neuron in the network.

The error of the network is given by:

Eq 3.2

$$E = \int_{t_o}^{t_1} q(\bar{y}(t), t)dt$$

t_0 marks the start of the correction interval and t_1 marks the end, with $0 \le t_0 < t_1 \le T_{total}$. The function q is the cost-function which depends on the activation of output neurons y^o and time t. With q for example the squared distance between the neuron output (y^o) and the desired output (d) (n is the number of outputs):

Eq 3.3

$$q(\bar{y}(t), t) = \sum_{i \in O} (y_i^o(t) - d_i(t))^2$$

3.2.1 Adjoint variable

The previous defined error function (Eq 3.2) needs to be minimized to a (sub-)optimum solution. The mathematical proof is however complex and can be found in Chapter 6.

The variable p_i which is introduced in the proof is called the adjoint variable which is introduced to constrain the system of equations for $y_i(t)$ in the learning process and is in this case defined as:

Eq 3.4

$$\frac{dp_i}{dt} = \frac{1}{T_i}p_i - e_i - \sum_{j=0}^{n} \frac{1}{T_j}\omega_{ij}F'(x_j)p_j$$

with e_i defined as:

Eq 3.5

$$e_i = \frac{\partial}{\partial y_i(t)} q(\bar{y}(t), t)$$

Eq 3.4 shows that the new p_i depends on the p_j, x_j and the weights between this neuron and the ***next*** neurons. Eq 3.5 shows how an error changes with the change of $y_i(t)$.

3.2.2 Adjustment functions

Now, learning of the network can be defined in the following way. First the weight update is considered (gradient descent-method):

Eq 3.6

$$\omega_{ij}^{k+1} = \omega_{ij}^{k} - \gamma \frac{\partial E}{\partial \omega_{ij}}$$

with:

Eq 3.7

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{1}{T_j} \int_{t_0}^{t_1} y_i(t) F'(x_j) p_j dt$$

The reader should note the dependency of the new weights on the outputs of the connected neurons.

In a similar fashion the update of the time constant can be given:

Eq 3.8

$$\frac{\partial E}{\partial T_i} = -\frac{1}{T_i} \int_{t_0}^{t_1} p_i \frac{dy_i}{dt} dt$$

The time constant is updated in the following way:

Eq 3.9
$$T_i^{k+1} = T_i^k - \gamma \frac{\partial E}{\partial T_i}$$

3.3 Discrete representation

The simulation of a neural network will be done on a digital computer. This implies that the equations in the continuous time model need to be approximated. The techniques used will introduce errors in the calculation of the differential equations. Where possible the Runga-Kutta method[9] is used in the calculations to minimise the calculation errors.

Eq 3.1 is the time-continuous model of the output. This equation can be rewritten to a discrete approximation using:

Eq 3.10
$$\frac{dy_i}{dt} \approx \frac{y_i(t + \Delta t) - y_i(t)}{\Delta t}$$

hence

Eq 3.11
$$y_i(t + \Delta t) = \left(1 - \frac{\Delta t}{T_i}\right) y_i(t) + \frac{\Delta t}{T_i} F(x_i(t)) + \frac{\Delta t}{T_i} I_i$$

and usually $y_i(0) = F(0) = 0.5$.

The discrete version of the error function (Eq 3.2), making use of the definition of the integral[1], is:

Eq 3.12
$$E = \Delta t \sum_{t=t_0}^{t_1} q(\bar{y}(t), t)$$

provided, of course, that the steps t makes are small enough (e.g. $\Delta t \to 0$).

The discrete version of the update rule for the weights is defined as:

Eq 3.13
$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\Delta t}{T_j} \sum_{t=t_0}^{t_1} y_i(t) F'(x_j) p_j$$

As with the weight update rule, it is now possible to derive the discrete approximation of the time constant update function:

Eq 3.14
$$\frac{\partial E}{\partial T_i} = -\frac{\Delta t}{T_i} \sum_{t=t_0}^{t_1} p_i(y_i(t + \Delta t) - y_i(t))$$

Some care has to be taken when $t_1 = T_{total}$, because at this point $y_i(t + \Delta t)$ does not exist.

The discrete approximation of the adjoint variable (Eq 3.4) is:

Eq 3.15
$$p_i(t) = \left(1 - \frac{\Delta t}{T_i}\right) p_i(t + \Delta t) + \Delta t e_i + \Delta t \sum_{j=0}^{n} \frac{1}{T_j} \omega_{ij} F'(x_j) p_j(t + \Delta t)$$

with:

Eq 3.16
$$e_i = \frac{q(\bar{y}(t) + \Delta y, t) - q(\bar{y}(t), t)}{\Delta y}$$

In the software implementation this approximation is replaced by the much more accurate Runga-Kutta method, because using Eq 3.16 produces an error which may result in undefined learning behaviour of the network: it is possible that the error of the network is almost the same as the error introduced by this (Eq 3.16) first order differential approximation.

---

1. The integral is defined as: $\int_{t_0}^{t_1} f(t)dt = \lim_{\Delta t \to 0} \sum_{t_0}^{t_1} (\Delta t \cdot f(t))$

The calculation of every $p_i(t)$ starts at time $t = t_1$ with $p_i(t_1) = 0$ (backpropagation over time).

4 Experiments

To investigate the capabilities and shortcomings of DRNNs some experiments were conducted. The setups and results of these experiment are described in this chapter.

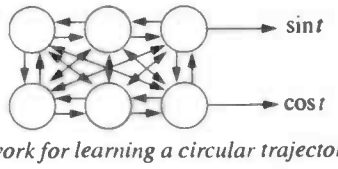
4.1 A circular trajectory

The setup of the first experiment is straightforward. A network of six neurons, four hidden and two output neurons, is constructed. There are no inputs besides the six bias inputs for every neuron. The target consist of a sine and a cosine, hence creating a circular trajectory. Two neurons of the network are appointed as output neurons: one should produce the sine and the other the cosine. Other settings of the network are:

Eq 4.1

$$\begin{aligned} learningrate &= 0.15 \\ \Delta y &= 0.01 \\ \Delta t &= 0.05 \\ bias &= 1.00 \end{aligned}$$

Figure 4.1



Structure of the network for learning a circular trajectory. As shown in this figure, no external inputs are present.

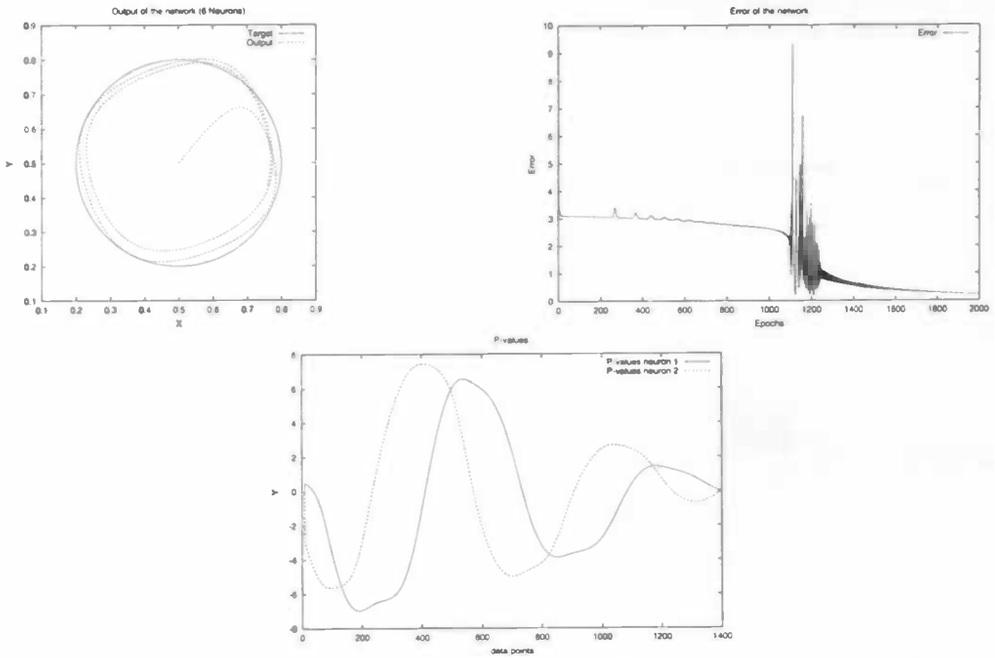
4.1.1 The results

Figure 4.2 shows the result of this experiment. Although the output does not match the target perfectly, it is clear from the results that the network can create a circular trajectory. To allow the network some time to start-up the p-value is not calculated for values at $t < 0.5$. In this short time period the network is allowed to get from $(0.5, 0.5)$ to the circle.

The error-plot shows a strange feature during learning of the network. After some time of gradual descent the error starts to fluctuate very violently before stabilizing again. This behaviour is predictable in the sense that every time a network is set to learn this task it shows a simular error plot. Draye [2] reports that in this time period (the so-called Hopf-phase) the network's structural stability is lost because of the chaotic nature of the network. During this period of time the error does fluctuate mainly due to the structural changes in the network.

This inherently instability of the network may seem a problem. However, only in a small number of cases the network stabilises on a higher error level, while most of the time it does reach a state that produces a small error value. The structural changes can even be a good thing: these 'wild' movements decrease the chance of getting stuck in a local minimum.

Figure 4.2

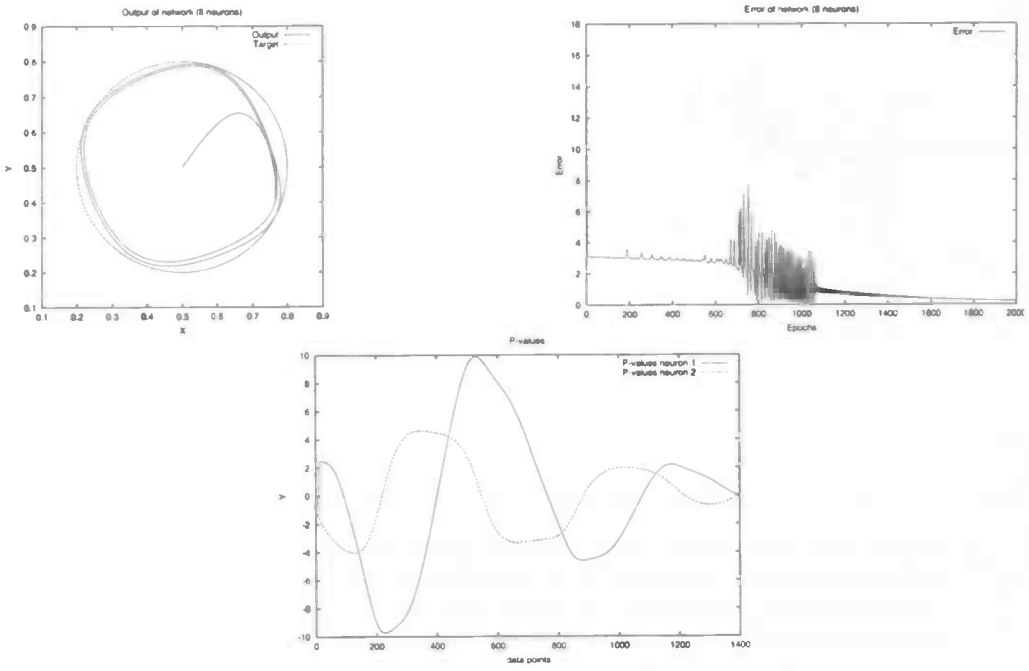


A circle created by a network consisting of six neurons. The upper-left image displays the target and the output after training, the upper right image shows the error of the network during learning and the image in the centre shows the p-values for the two output neurons. The length of the target is about two periods, in other words: the network circles around two times.

4.1.2 Adding more neurons

As shown in the previous section a network with six neurons can create a circular trajectory. But what happens if more neurons are added to the network? Figure 4.3 shows the outcome of this new experiment.

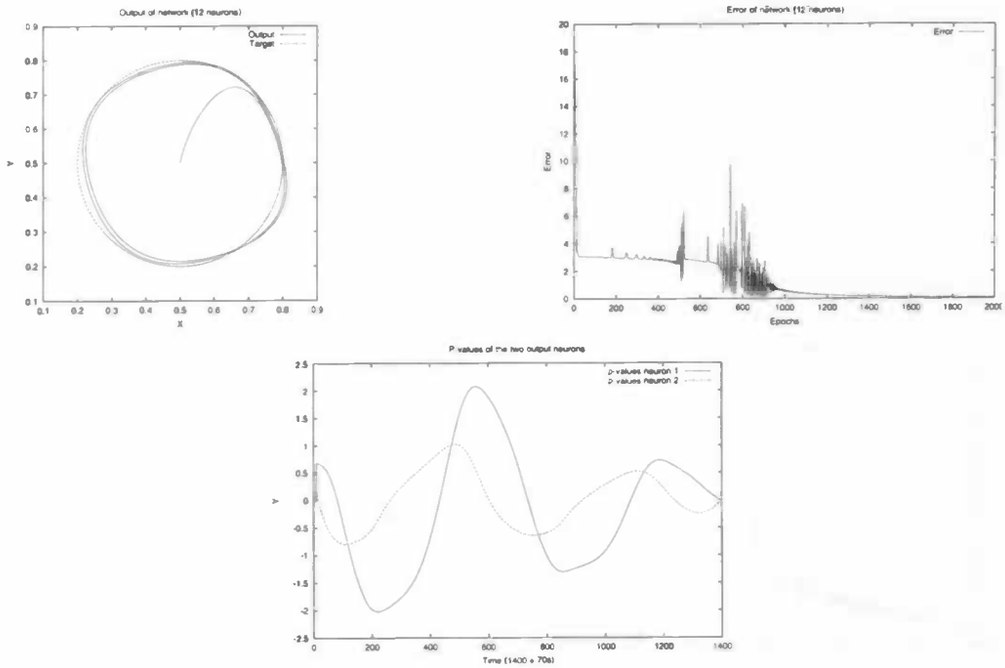
Figure 4.3



A circle created by a network consisting of eight neurons. The upper-left image displays the target and the ouput after training, the upper right images shows the error of the network during learning and the image in the centre shows the p-values for the two output neurons. The length of the target is about two periods, in other words: the network circles around two times.

Figure 4.4 shows the results for a network with twelve neurons. Although the end result matches that of the network with eight neurons, the p-values are significantly smaller.

Figure 4.4

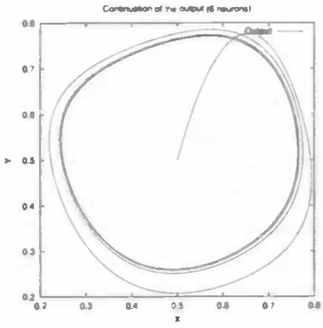


A circle created by a network consisting of twelve neurons. The upper-left image displays the target and the output after training, the upper right image shows the error of the network during learning and the image in the centre shows the p-values for the two output neurons. The length of the target is about two periods, in other words: the network circles around two times.

4.1.3 Continuation of the signal

What happens to the output of the network if the time period is extended? Will the behaviour continue or will it become unpredictable or even unstable? Figure 4.5 shows that the network (consisting of six neurons) seems to produce a limit cycle that continues even after the learning period. This result shows that even a few neurons can be responsible for cyclic behaviour.

Figure 4.5

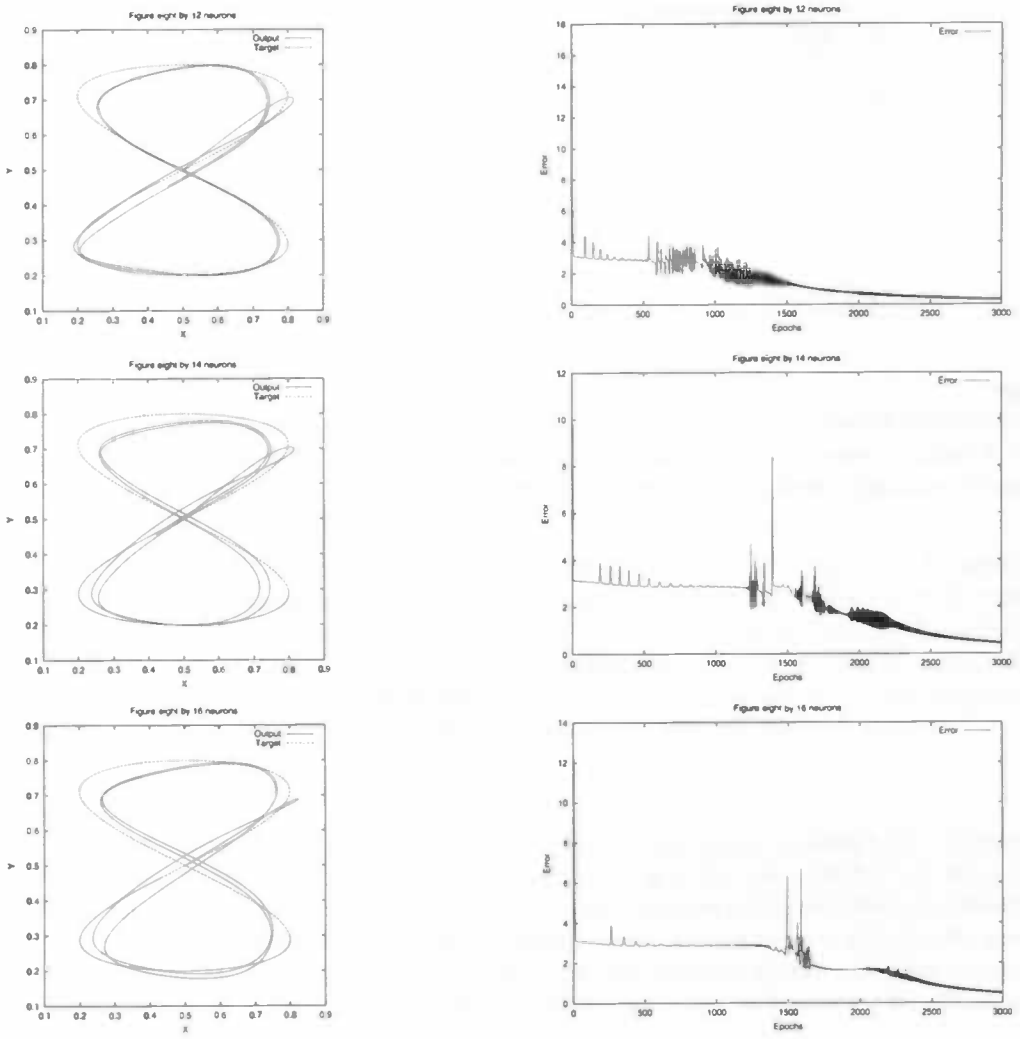


Continuation of the network with six neurons after the learning period.

4.2 Figure eight

A figure eight, or Lissajous Curve, can be created by speeding up one of the components of a circle. To be exact: double the current frequency of that component. In Figure 4.6 the results are shown.

Figure 4.6



Outputs and errors for twelve, fourteen and sixteen neurons.

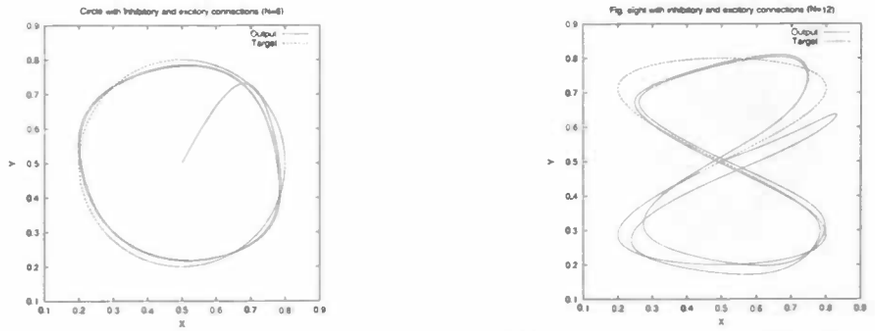
The network requires at least twelve neurons to learn the task. Other than that, the results are similar to those of the circle.

4.3 Effects of inhibition and excitation

A biological neural network consists of neurons, axons, dendrites and synapses[5]. A signal from one neuron travels through the axon via the synapse onto a dendrite or the cell body of the receiving neuron. In general, every connection between a transmitting neuron and its receivers are either all inhibitory or all excitatory. Which one it will be depends on the type of neurotransmitter the neuron produces. An inhibitory connection can be seen as a negatively weighted connection and an excitatory connection as a positively weighted connection.

It is interesting to see what the effects of such a biological limitation are on the overall behaviour and learnibility of a neural network. In the following experiment the circle and figure eight are again used as targets for the network. However, this time the neurons are set to be either excitatory or inhibitory with a 50% distribution (meaning that every neuron has 50/50 chance of being either excitatory or inhibitory). In Figure 4.7 the results are shown.

Figure 4.7



The results of the training of a network with inhibitory and excitory connections.

The network seems to have no trouble learning the circle. The output of the network matches the target within a small margin of error. When compared with the results as shown in section 4.1, the results are simular and one could conclude that in the case of the circle the constrains on the network have little or no effect.

With the figure eight the result is good: the network can produce the correct behaviour. However, the deviations from the target are much larger than the deviations shown in Figure 4.6. In this case the network is somewhat limited by the inhibitory and excitory connections. One could conclude that with more complex tasks a constrained network will have more and more difficulties in comparison to an unconstrained network. However, these two expriments give not enough evidence to support this claim, therefore more research is required to see if this is indeed the case.

4.4 The effects of noise

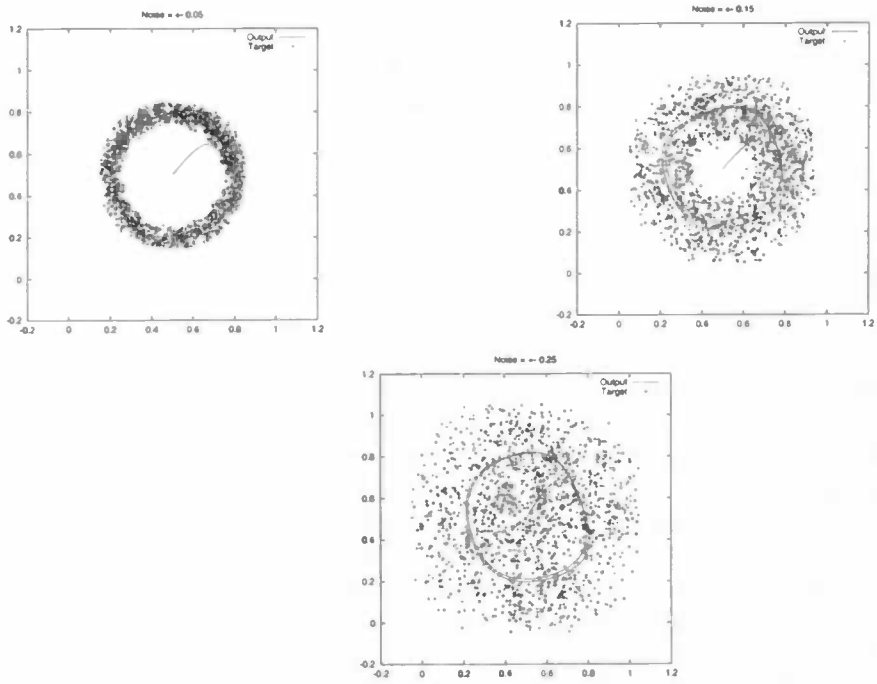
In practice the targets of a neural network are hardly ever without noise. Although the following experiment is not very complex, it gives some understanding of the way a DRNN will react to noise. In this setup the circle is used. The targets are now distorted with a uniformly distributed amplitude noise. This means that at every time index a small random value is added to the sine and cosine functions. This way the global trend of the sine is always visible, but when the two components are used to plot the circle it shows a malformed circle. In Figure 4.8 the effects of this distortion can be observed.

The result of this experiment looks very promising. Even with a noise of ± 0.25 (note that the range of the sinus used is $[0.2, 0.8]$) the network has no trouble at all to reach the limit cycle of a circle. However, from a mathematical point of view these results are not suprising. Chapter 3 explains that the integral of the p-value is used in calculating the energy E of the network. The p-value contains the error-value of the network for the case where the neuron is an output neuron (Eq 3.15). In this experiment, the error-value consists of the normal error with some random value a. When the distribution is uniform and $a \in [-b, b]$ then with a large enough amount of random values the integral cancels out the effect of the noise (Eq 4.2). One might say a DRNN has an built in filter for noise with zero mean.

Eq 4.2

$$\int \sin t\,dt \approx \int (a_t + \sin t)\,dt$$

Figure 4.8



A circle with respectively ±0.05 , ±0.15 *and* ±0.25 *uniformly distributed noise. The targets are build using:* $y(t) = a + \sin t$ *with* a *a random variable with the given distribution.*

4.5 XOR

The first artificial neural networks were developed in 1943 by McCulloch and Pitts[8]. These primitive neurons were able to solve simple problems like OR and NOT and so on. In 1956 Rochester created the first computer simulation of an ANN. From that point on some very interesting research was conducted. But at the end of the nineteen-sixties Minsky and Papert published an influential paper describing a number of problems that were unsolvable for perceptrons of that time. One of these problems was the XOR: this logical function is nonlinear and perceptrons can only solve linear problems. After this, ANN research was abandoned by most researchers. It was not until the development of the multilayer feedforward perceptrons that the XOR-problem could be solved by an artificial neural network. So, historically, the XOR-problem is viewed as *the* testcase for new types of ANNs. If a network is unable to solve this relatively simple problem, it is not a very realistic type of ANN.

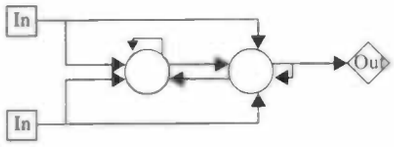
Before an experiment is set up with a DRNN, an observation needs to be made. A DRNN is a network designed to learn time-dependent problems. A logical function like the XOR is binary and has no time information what-so-ever. So, instead of a single '1' or '0' a time signal had to be created (Eq 4.4), hence lists of either ones or zeros are used as time signals. These signals are then used as inputs and targets for the network. The network is again allowed some start-up time before it has to match the target value. The parameters for the network are:

Eq 4.3

$$\begin{aligned} learningrate &= 0.15 \\ \Delta y &= 0.01 \\ \Delta t &= 0.01 \\ bias &= 1.00 \end{aligned}$$

The structure of the network is shown in Figure 4.9.

Figure 4.9



The structure of the network for the XOR-problem.

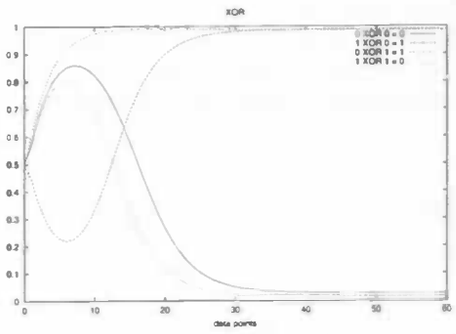
Eq 4.4

$$\begin{array}{ll} \text{one} & y(t) = 1.0 \quad \text{for: } 0 \le t \le 1400 \\ \text{zero} & y(t) = 0.0 \quad \text{for: } 0 \le t \le 1400 \end{array}$$

Figure 4.10 shows the outputs of the network. Although the output never reaches one or zero, the classification is a success (the logic state '1' is representated by a real value close to one and '0' is representated by a real value close to zero). In this picture the start-up is evident and quite particular: in three of the four cases it first tends towards the wrong value before stabilizising to the correct value. Some other observations were made during this experiment:

- Adding more neurons will not give much better results
- The best results are achieved when presenting the input-output pairs in a random order. Using a predefined order requires a somewhat longer training time. Training one particular pair (priming) and after a while adding the rest to the training gives even worse results
- Disabling learning of T (hence fixing it to $T = 1$) produces a somewhat shorter training time, but the end result is the same
- Setting the bias to either 1 or 0.5 does not really matter

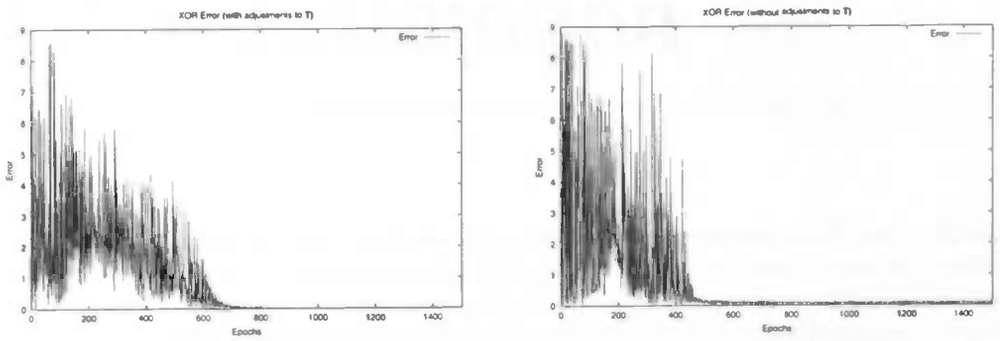
Figure 4.10



The output of a network of two neurons after training of the XOR problem.

The error during training is depicted in Figure 4.11. These results match the results of Pearlmutter, who also conducted experiments with DRNNs. Pearlmutter had the same startup effects and the network he created also needed about 1000 epochs to learn the given task.

Figure 4.11



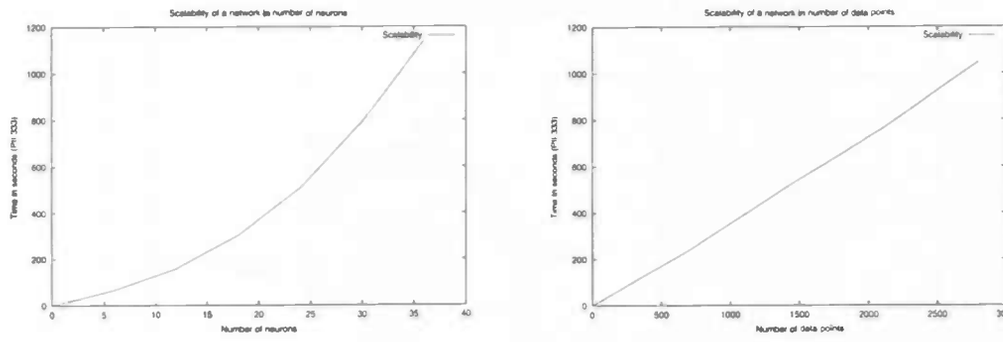
Error during training of a network of two neurons. The left image gives the error when the time constant T *is adjusted during training and the right image show the error when the time constant is set to* 1 *and kept fixed during training.*

4.6 Scalability

The experiments run so far were conducted with a small number of neurons. But can this implementation be used to simulate networks with a large number of neurons? In (most) practical applications dozens of neurons may be required to learn the given problems. To see the effects of adding neurons to the network the experiment of the circle was rerun with different network sizes. Figure 4.12 (left) shows that the amount of time needed to complete a fixed number of epochs grows approximately quadratic. This is disastrous for very large networks. It also means that adding more (single processor) computing power will not really help. There is one upside to this result: because the number of connections in a network is n^2 (hence by adding one neuron the network needs $2n + 1$ new connections), the computational time grows with the same rate.

The solution might be to implement DRNN's on multiprocessor systems. For example, if a network is simulated on a system using four processors the time needed will decrease by a factor four. With current developments, however, the future will bring multi-layered (optic) systems with many small processors which will be very suitable for the simulation of very large networks.

Figure 4.12



Scalability of the implementation when adding neurons (left) or adding data points (right).

On the right side of Figure 4.12, the scalability of the network is plotted when adding data points to the target and keeping the number of neurons constant. As expected, the time needed increases linearly.

5 Pitch Estimation

This chapter describes the pro's and cons of using dynamic recurrent neural networks in a sub-task of the speech recognition process: pitch estimation. In the first sections the problem and setup of this experiment are discussed. After that, the results are shown followed by some more general remarks. Information on the cochlea model will be restricted due to patent rights and other confidential information.

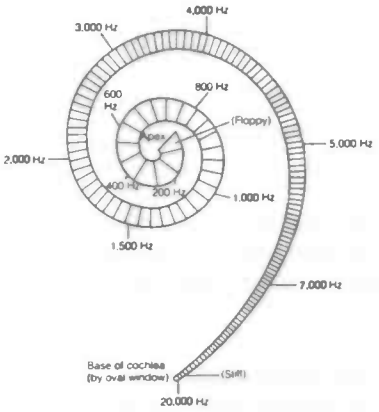
5.1 Model of the cochlea

Usually, speech signals are analysed by creating a fourier spectrum or other similar mathematical transformations. These methods have several disadvantages, two of which are:

- a transformation window size has to be chosen. Most of the time the size is fixed, but in any case the problem is similar to that of the chicken and the egg: to be able to perform a good recognition an optimum window size has to be chosen. But in doing so, one needs to know to some extend what is said. For example, the chosen window should overlap an utterance of a 'six' and not divide the utterance into several parts. But if you don't know where the utterance starts and ends (in other words: you don't know what is said) it is very hard to select the correct window size and place it at the correct time index
- any phase information is lost: speech recognition system designers usually take the amplitude plot of the FFT, discarding the phase plot

Another way of performing a preprocessing algorithm is based on the way the cochlea in the human ear functions. In short, the cochlea gives not only an amplitude response, it also contains vital phase information (hence its response preserves time information). For each frequency component in a given sound a different part of the basilar membrane in the cochlea responses. Hence the human auditory sensor can sense which frequency was detected at a particular point in time, thus combining phase and amplitude information. See Figure 5.1 and [5].

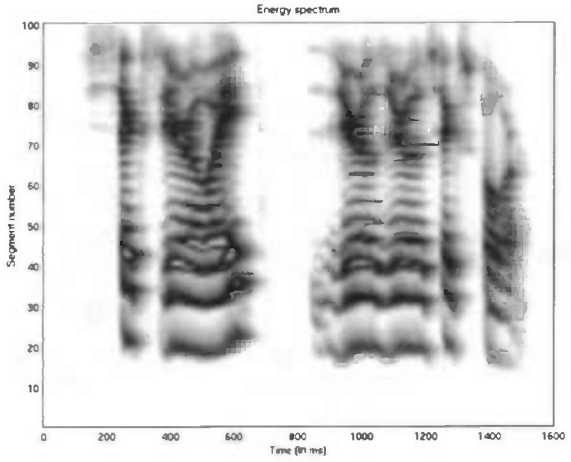
Figure 5.1



The basilar membrane of the human cochlea. High-frequency sounds produce their maximum displacement near the base. Low-frequency sounds produce their maximum displacement near the apex. From [5].

The artificial cochlea is divided into segments, with each segment maximum responsive to one single frequency. A spectrum plot is created by summing the output values per segment (displament values) over a certain time period (for example 5 milliseconds). See Figure 5.2.

Figure 5.2



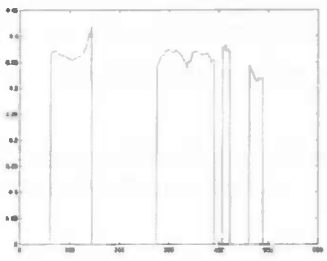
The cochlea energy spectrum for the utterance 'six zero eight zero zero six four'. Segment number 100 is maximum responsive for about 4 kHz and segment number 0 for about 25 Hz. The segment-frequency map is approximately logarithmic.

5.2 Pitch estimation

If the utterance contains voiced speech, it (nearly) always has a pitch. This means that voiced speech is built up by one frequency (the pitch) with several harmonic frequencies. If one takes a closer look at the spectrum at a particular time, one could give an estimate of the pitch. This process has already been automated[4], and produces a plot like Figure 5.3.

The current algorithm is not fail-safe. The output is sometimes distorted by what is called an octave-error: the estimate is exactly one octave above or below the correct value. Figure 5.3 shows another mistake by this estimation process. The first utterance of 'six' is missed. The first type of mistake can be relatively easily detected and removed from the training set, the second type is somewhat harder to detect (and has not been removed from the training set).

Figure 5.3



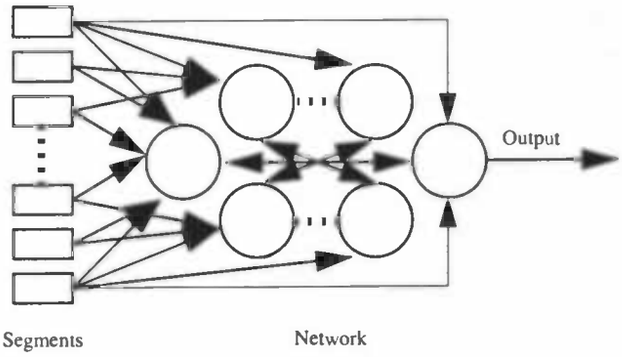
Pitch estimate for 'six zero eight zero zero six four'. The minimum pitch is set to 0.15 for the training later on. The maximum pitch is set to 0.85, also for training purposes. These limits correspond to a range of a 45 to 400 Hz pitch. Hence the y-axis is the scaled log of the pitch frequency.

Although the training set is distorted, the network may be able to generalize enough to avoid making the same mistakes. The errors in the training set also present another problem: is it possible to say the network has learned to give a good pitch estimate and if so, why? For a further discussion of these and other problems see section 5.5.

For this task a neural network is constructed with sixty neurons. Only the lower sixty-three segments of the cochlea are used (corresponding to a 20 to 2000 Hz frequency range), because only these contain enough resolution to see the different sub-harmonic components of the pitch. All neurons receive inputs from the segments. This means that every neurons has access to the infor-

mation of the complete cochlea over time. One of the neurons is used for output, hence it too is connected to every segment.

Figure 5.4



Network lay-out. Not all neurons, inputs and connections are shown, because the picture would then become a single black spot on the page.

5.3 Time frame and learning rate

The parameters for the network are set to:

$$\eta = 0.01$$
$$\Delta t = 0.005$$
$$\Delta y = 0.01$$
$$bias = 1.00$$

The cochlea uses a frame length of 5 milliseconds for each energy value, hence the $\Delta t = 0.005$. The internal sample frequency for each of the segments is about 16 kHz, giving each frame a length of about 80 samples (hence each frame consists of about 80 displacement values).

5.4 Training and validation sets

The training set consists of about 200 different (random) utterances from male (about 130) and female (about 70) speakers with a 2.75 seconds length. This gives an input data-stream of 550 points (5 milliseconds per frame) per segment. For each of these utterances a pitch estimate is calculated which is used as the target for the output neuron.

The validation set consists of four examples from the training set and two new examples, all female.

The estimated pitch (the target, see Figure 5.3) contains two types of disturbances that are removed by using a low-pass Kaiser-Bessel filter:

- small, local, errors
- discontinuations

Both disturbances introduce high frequency elements to the target signal. Because a Kaiser-Bessel filter removes these high frequencies a much smoother target signal is created. When these effects were removed from the targets, the network seemed to be able to learn the given task much quicker and it performed better after training.

5.5 Results

In the next section the performance of the network in situations where the algorithm functions adequate will be discussed. The dataset consisted of four trained and two untrained examples. After this first experiment the trained network is presented data with which the current estimator had trouble giving good estimates: some octave errors are present, as well as other errors in judgement. It is also of interest to see the effects of missing data on the network: how will it perform and can the results give an indication of how the network uses the available data? These questions are discussed in the final section.

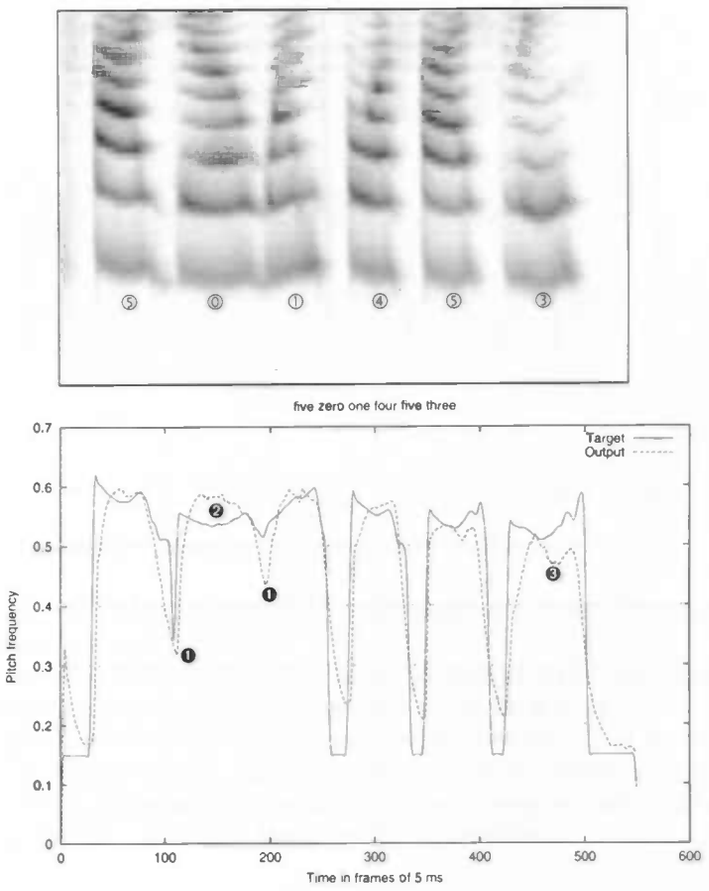
5.5.1 Results with error-free data

In this section the results are discussed when the training of the network was finished. During training the output of the network was calculated for a randomly selected input and compared to the target estimation. With the resulting error the weights of the network were adapted. This process continued until the network did not show any significant improvements between several epochs. The following four spectra were part of the training, but the last two spectra presented in this section were part of the test set: the network did not encounter them during training.

When a particular spectrum and its pitch estimation are discussed generalized conclusion are made based on a single image. This is scientificly unsound, of course, therefore only conclusions and comments are noted that are more or less general remarks: evidence for the stated claims can also be found in other cases. If a claim is only evident in the referenced figure, it is usually because either the network or the estimator made a clear mistake, for example an octave error.

In the spectra an utterance is denoted by the number in a circle (e.g. ①). However, in english two different ways of pronouncing the zero are possible: 'o' (or 'O') and 'zero', therefore the first is denoted by the ⓪ and the latter by the ⓪. In the pitch estimation plots interesting points are numbered from ❶ to ❾. The numbered comments in the text refer to these points in a particular plot.

Figure 5.5



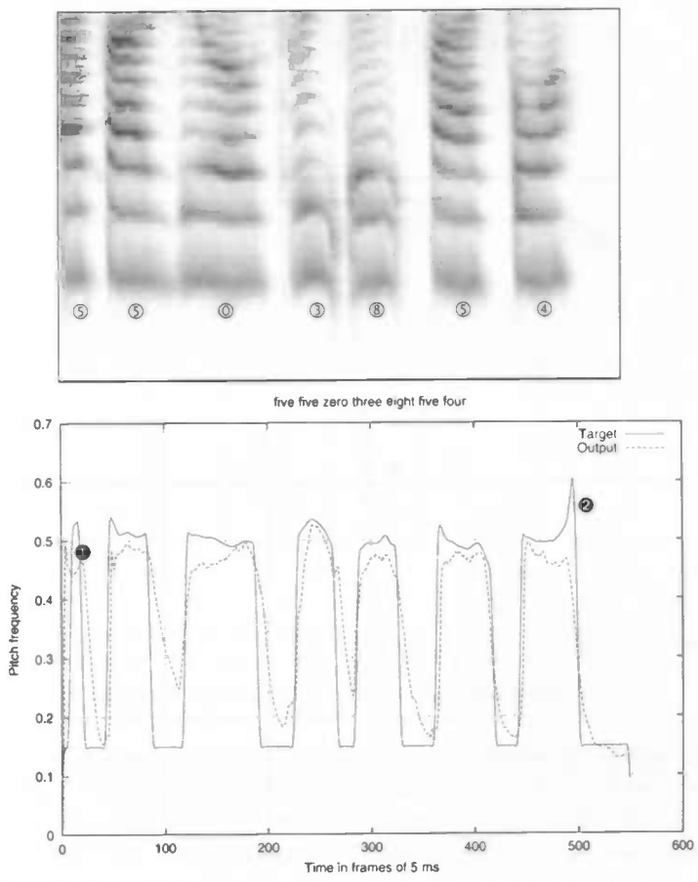
Cochleagram and the pitch estimations for 'five zero one four five three'

Figure 5.5 shows promising results. Because a spectrum is highly dynamic and the target is the result of an estimation process, a perfect match is nearly impossible. However, in this case the network is very close to the target. The on- and offsets for the target and the network are equally timed and the estimated pitch is a close match. Three, more specific, observation are:

1. The network seems to have a better on- and offset detection than the target algorithm: when two separate utterances (or when a single utterance contains more then one voiced parts like 'seven') have little temporal space between them, the network detects them as being two events while the estimator is prone to make mistakes during the transition from one voiced part to the other.

2. If pitch of the first 'five', 'zero' and 'one' are compared, the spectrum shows a smaller pitch value for the 'zero'. However, the network gives a pitch estimate similar to that of the 'five' and 'one': this seems to be too high.
3. Although it is hard to see whether the network is slightly off or not, the output follows the contour of the pitch quite nicely. In the spectrum the 'three' has a significant drop in the pitch half way, which is also noticeable in the networks estimation.

Figure 5.6

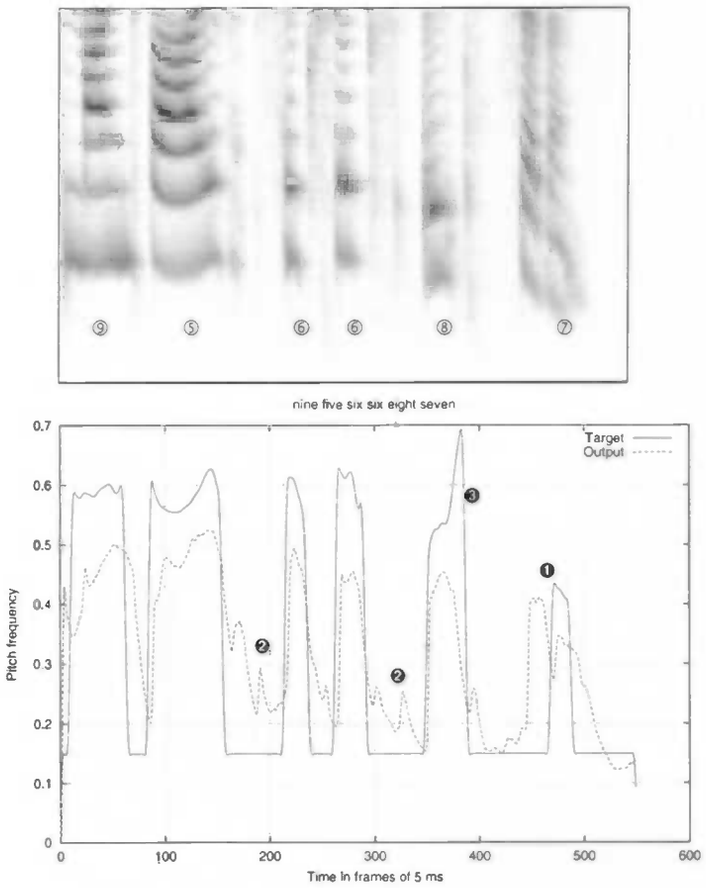


Cochleagram and pitch estimation for 'five five zero three eight five four'

Again, like in Figure 5.5, the output of the network in Figure 5.6 is close to the target pitch estimation. There are just two small notes:

1. The start-up of the network prohibits a good estimate. Usually the first frames do not contain any voiced speech but in this case they do: therefore it seems the network tries to divide the event into two separate parts but the first peak is the result of the start-up of the network.
2. With the estimation for the 'four' the original algorithm makes an obvious mistake at the end of the utterance. The spectrum and the network both show a nearly straight pitch but the algorithm produces an estimate which rises from about 0.5 to about 0.6.

Figure 5.7

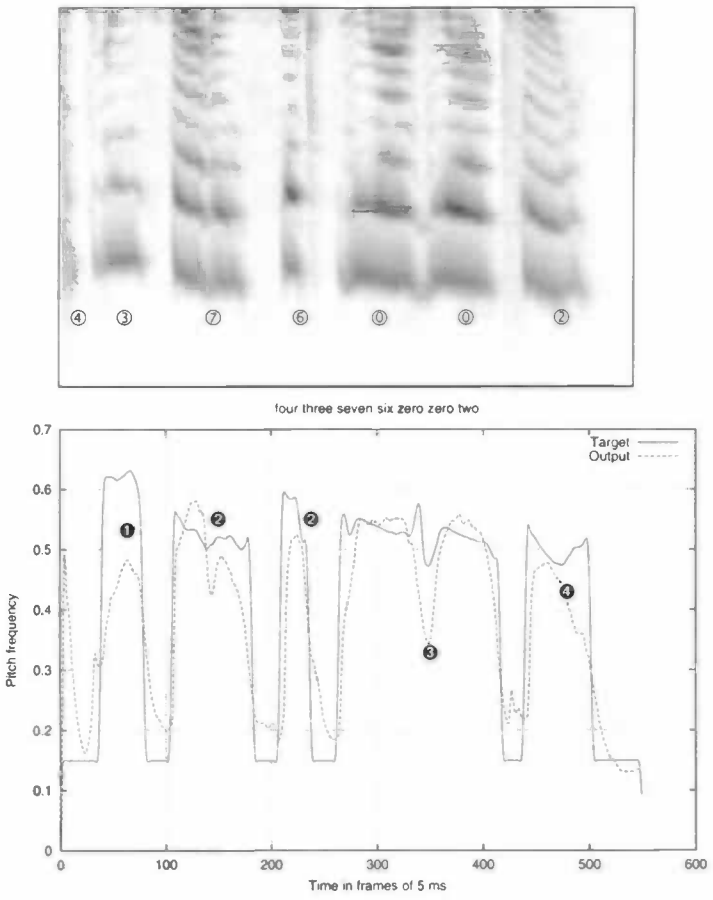


Cochleagram and pitch estimations for 'nine five six six eight seven'

Although the pitch flows in the target and the output of the network are similar, the estimate values are very different. The original algorithm misses the first voiced part of the 'seven' even completely (point 1). If the spectrum in Figure 5.7 is compared to that in Figure 5.6 and also taking the closeness of the pitches in the spectrum in Figure 5.7 into account, one could conclude that the network is closer to the truth than the estimator.

1. As stated before, at this point the original estimator misses the first voiced part of the 'seven'.
2. The network seems to give a response to the 's'-sounds. An 's' is unvoiced: the on- and offset detection of the network gives a response where it should not have.
3. Like in Figure 5.6 the original estimator makes a big mistake in the estimation.

Figure 5.8

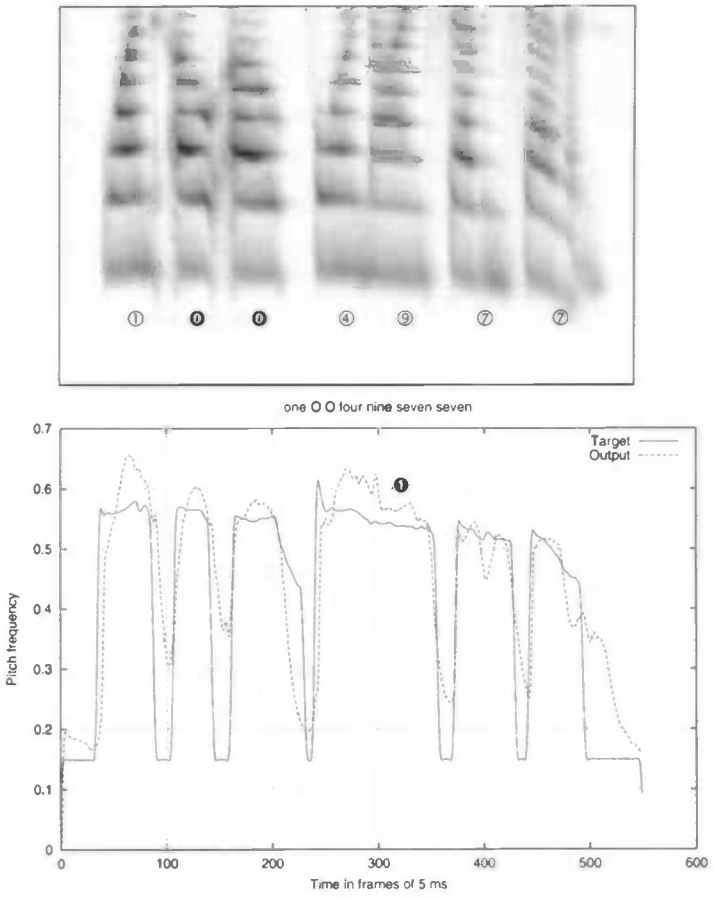


Cochleagram and pitch estimations for 'four three seven six zero zero two'

In Figure 5.8 the estimator and the network are for most part close together, except in the following cases:

1. The network is too low with its estimated pitch value. It is clear that the pitch of the 'four' is higher than the pitch of the 'three': the original estimator is probably much closer to the true pitch value.
2. In these cases it is much harder to conclude which estimate is correct. They are only about 15 Hz apart and because the spectrum is also log-scaled one can not dismiss either one.
3. The network shows a much better voiced event separation than the estimator. The latter shows fluctuations in its estimate were evidence for a pitch is missing.
4. This particular point shows a interesting response of the network. It seems the network uses the upper segments for the pitch flow estimate: although the energies in the lower frequencies indicate a drop in the middle of the voiced event (which is also shown by the estimator), the energies in the higher frequencies show a flow downwards from about one third of the event. The response of the network shows the same flow as the energies in the higher frequencies. This claim is supported by the results of the experiments as described in section 5.5.3.

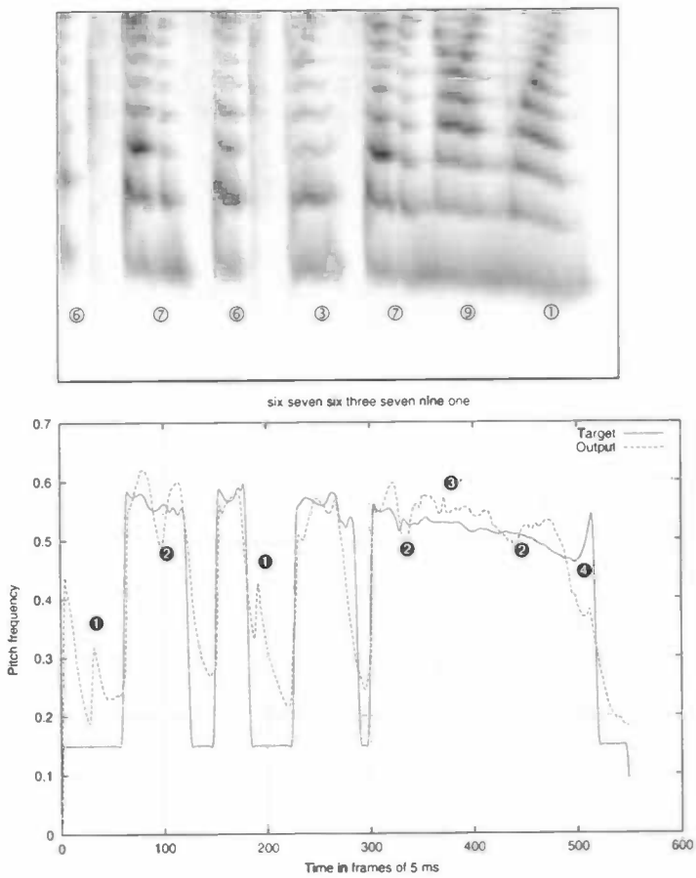
Figure 5.9



Cochleagram and pitch estimations for 'one O O four nine seven seven'

The spectrum in Figure 5.9 is part of the test set. Again it is difficult to see which of the estimates is correct (like in point 1), but it is obvious the network has generalized enough to be able to cope with completely new data. The network did not learn the (relatively small) data set 'by heart' (so-called over-specification), but it did indeed learn a pitch estimation algorithm. The demonstrated spatial and temporal generalization is impressive, especially when one considers the network only consists of sixty neurons.

Figure 5.10



Cochleagram and pitch estimations for 'six seven six three seven nine one'

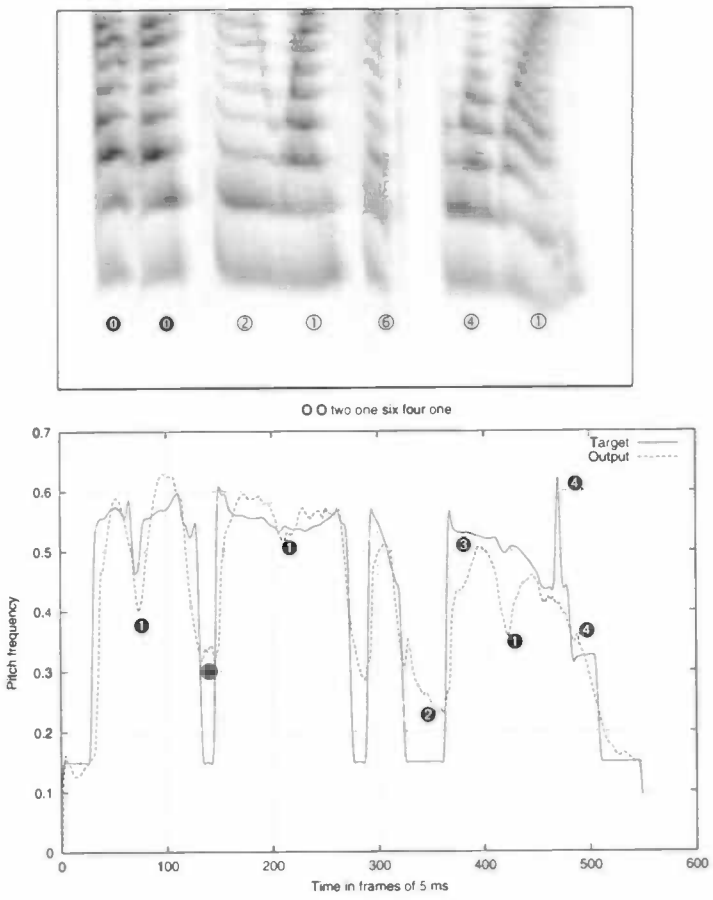
Figure 5.10 also shows a spectrum from the test set. Both systems produce a similar plot. The interesting points in the signal are:

1. Responses of the network to 's'-sounds. Although an 's'-sound is part of natural speech, the network should not have given any output.
2. The network can also identify separate but close voiced events in the test data.
3. Here a separation should be visible: the 'seven' ends about here and the 'nine' starts from about this point. But the spectrum shows one of the most difficult effects in speech for speech recognition systems: the 'seven' ends with the same sound as the 'nine' starts with, hence the speaker connects the separate events to one event. This makes it look as if one long event occurred.
4. A large error in the target estimation.

5.5.2 Trained network with more difficult data

In the following section a trained network is confronted with new data. In itself this is not very different from the last two examples in section 5.5.1. But the six selected input-output pairs contain errors in the estimations, so they were initially removed from the data set because they interfered during training which then produced unstable networks. But if the spectra are presented to the network, the produced output should give an indication whether the network generalised enough or it learned the exact same algorithm as the algorithm that produced the target.

Figure 5.11

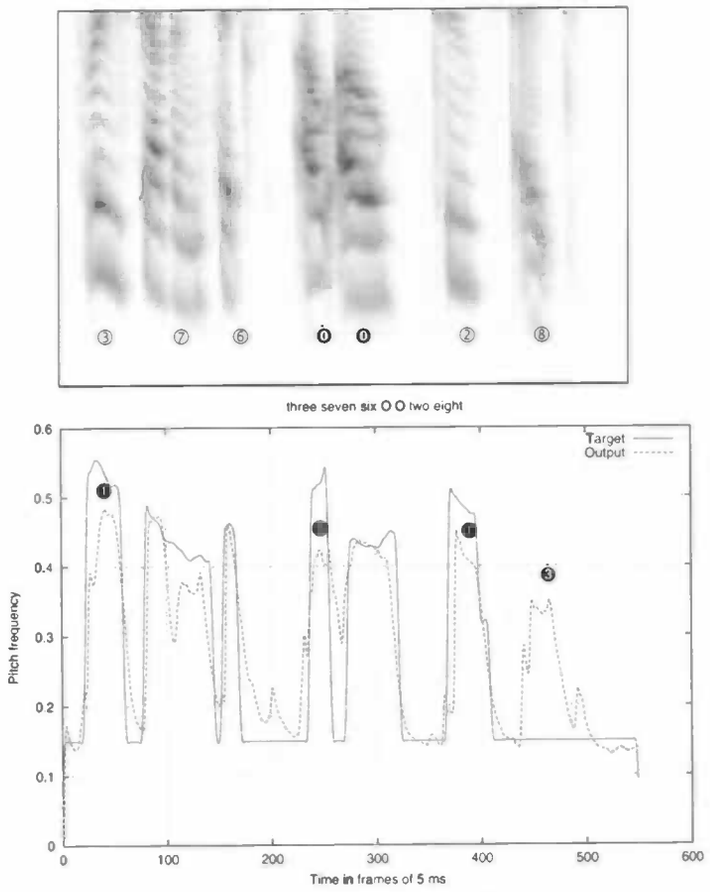


Cochleagram and pitch estimations for 'O O two one six four one'

Both the network and the estimator make mistakes in Figure 5.11, but the overall performance of both processes is comparable:

1. Here the network makes a much better separation between two distinct voiced events. However,
2. at these points the network generates a significant output where the spectrum shows no voiced speech, only noise or unvoiced speech.
3. The spectrum shows clearly a pitch, although the upper segments lack enough energy. The network misses this first part of the event.
4. At the end of the 'one' the estimator has trouble determining the correct value. The first peak is clearly an octave too high, but the second error is not an easy one: the upper segments show a downward flow of the energies but the first component remains at the same height. As stated before, the network usually follows the upper segments, but the estimator has some difficulties with the spectrum at this point.

Figure 5.12

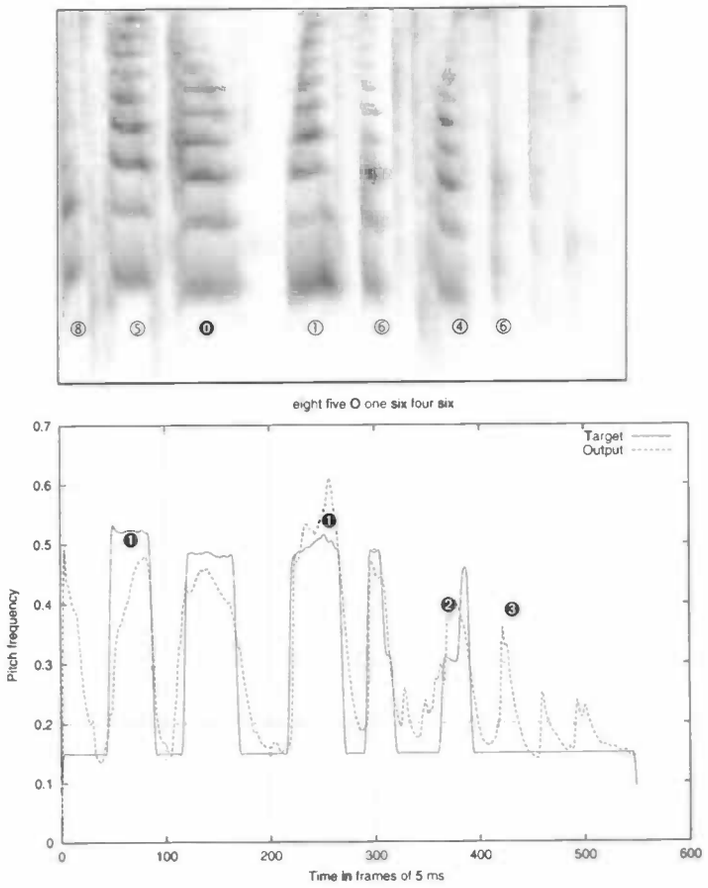


Cochleagram and pitch estimation for 'three seven six O O two eight'

In Figure 5.12 the network responds much less to energy of noise or unvoiced speech. The difference between the output by the network and the target is quite large, in contrast to Figure 5.11:

1. In estimating the pitch for the 'three' the network seems to be too low. The pitch is higher than the pitch of the first part of the 'seven', but the network responds with the same estimation.
2. This is hard to judge. The energy in the upper segments is distorted but the first component in the spectrum indicates a higher pitch than the second 'O'. However, both utterances are 'O', which could lead to the conclusion that the network's response is better. Both are probably incorrect and the real pitch is somewhere between 0.41 and 0.53.
3. Here the estimator misses the 'eight' completely.

Figure 5.13

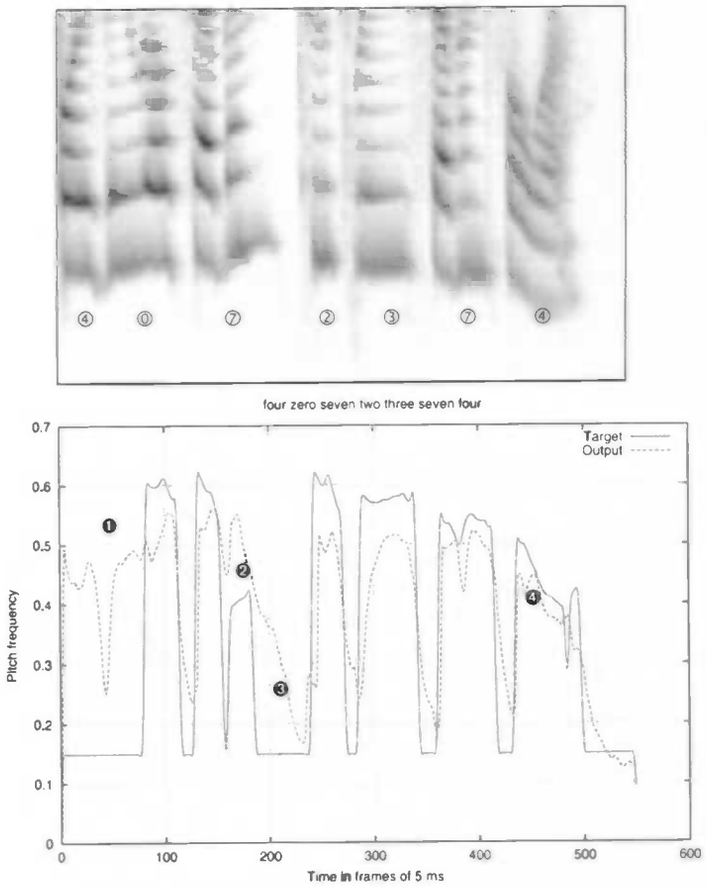


Cochleagram and pitch estimations for 'eight five O one six four six'

There are three interesting points in Figure 5.13:

1. At these points the network makes an incorrect estimation. In the case of the 'eight' for example, the pitch is somewhat higher than the pitch of the 'five'.
2. A clear octave error by the estimator.
3. Probably the first peak comes from the voiced part of the 'six', but the other peaks are responses to noise and are therefore mistakes. But on the other hand, the estimator misses the 'six' completely.

Figure 5.14

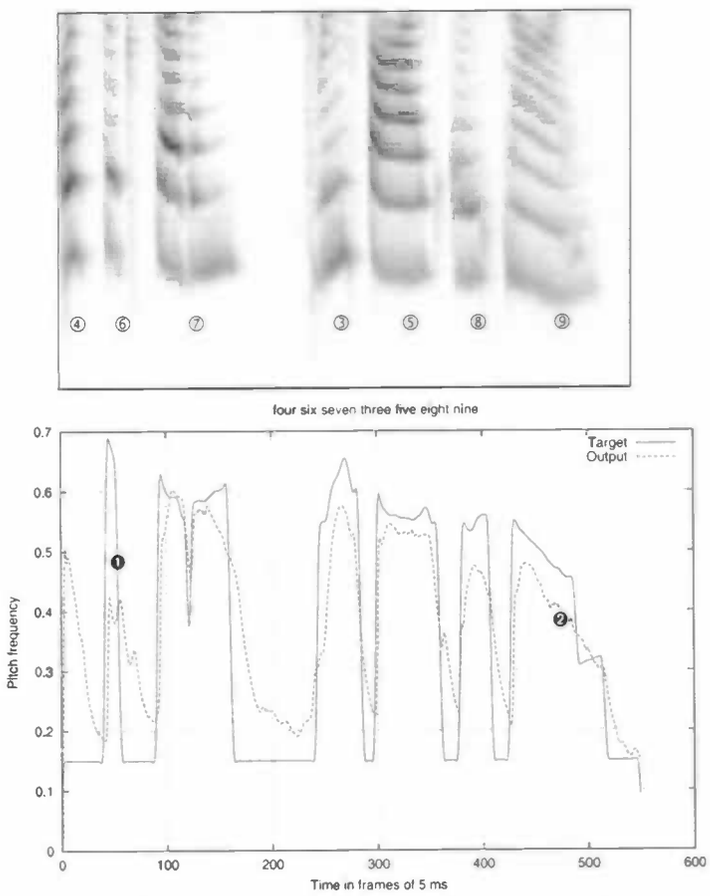


Cochleagram and pitch estimations for 'four zero seven two three seven four'

In Figure 5.14:

1. The estimator completely ignores the voiced part of the 'four' and most of the 'zero'.
2. At this point the correct estimation is most likely the output of the network: the second part of the 'seven' has clearly the same pitch as the first part.
3. The network continues with the pitch while only the first component has any energy left: another indication that the upper segments are used by the network for the value estimate and the lower segments for the on- and offset detection.
4. For most part of the final 'four' both the target and the network follow the pitch contour, although the estimator makes a short octave error at the end.

Figure 5.15

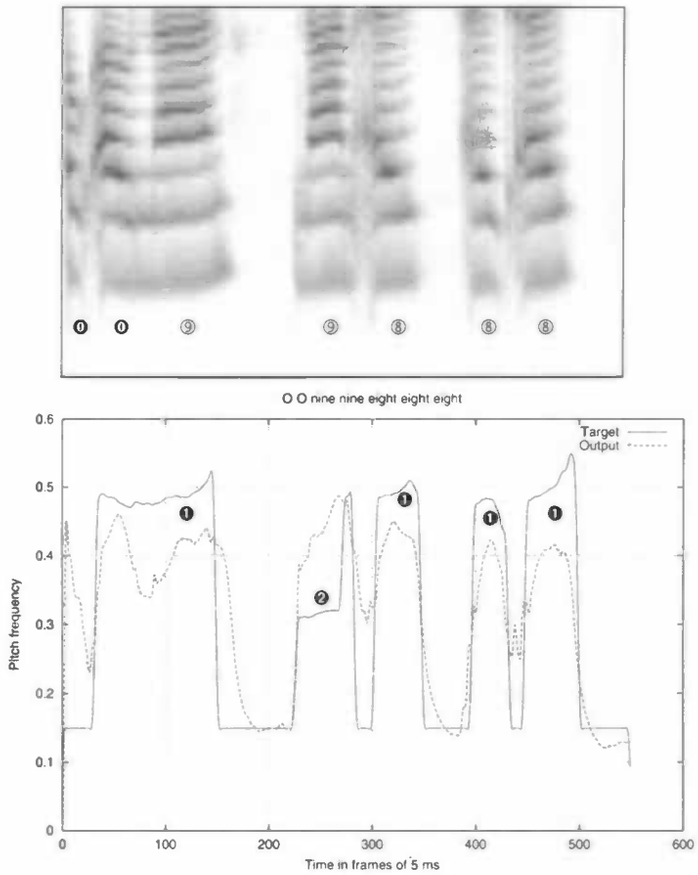


Cochleagram and pitch estimations for four six seven three five eight nine'

In Figure 5.15:

1. This is the only documented case where the network makes an octave error. This most likely comes from a lack of energy in the upper segments: it is present but it is not much.
2. A very nice pitch contour.

Figure 5.16



Cochleagram and pitch estimations for 'O O nine nine eight eight eight'

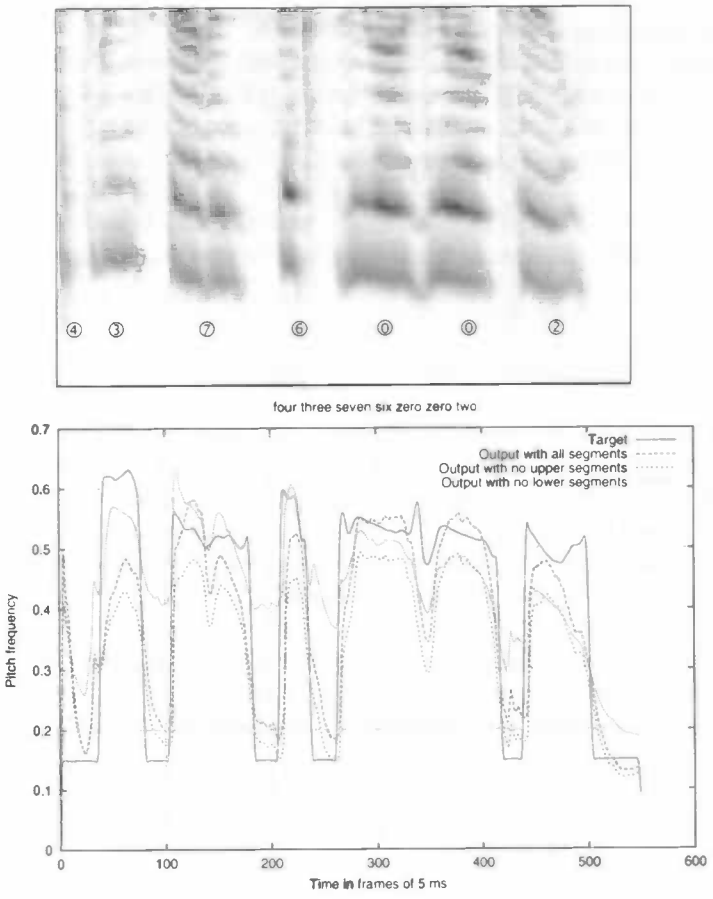
In Figure 5.16:

1. At these points the network is probably too low in its estimation.
2. An octave error by the estimator.

5.5.3 Response of the network when the upper or lower segments are removed

In the previous two sections a few remarks were made concerning the relevancy of the first component and the harmonics to the network. To get more insight in how the network deals with these two separate elements two experiments were run. In Figure 5.17 the results of these two experiments are shown. First the network was presented with a spectrum with the lower twenty segments removed (=set to zero), effectively removing (most part) of the first component. In the second experiment the information in the upper twenty segments was removed, hence removing most part of the harmonics. Figure 5.17 confirms the remarks made in section 5.5.2: with the upper segment intact, the network still performs reasonably well in the estimation of the pitch, but the on- and offset detection is distorted. On the other hand, when the information in the lower segments is still present, the on- and offset detection functions even better, but the pitch estimation is off target most of the time.

Figure 5.17



Cochleagram (with all segments showing) and pitch estimations for 'four three seven six zero zero two'

5.6 General remarks

The network responds very well after training. The on- and offset detection of the network is even better than the target estimation process. And most of the time the trained network comes also very close to the target estimated pitch. However, it is hard to say which estimation process is better: the intrinsic dynamics of speech and (small elements of) noise in the signal make it nearly impossible to see which process is best. The only real way of finding out the truth, is by using both estimation processes in speech recognition and see which one produces the best results.

The experiments in this chapter have shown that a DRNN has powerful temporal and spatial generalisation potential. A DRNN can therefore be a strong tool even when the presented inputs and targets are highly complex and dynamic!

However, the training of a DRNN is supervised: for each input there is a known target output. Usually these targets are perfect, although sometimes the targets are distorted by noise. The targets in the pitch estimation are not perfect: the current pitch estimation algorithm does produce errors. If these errors are severe enough, the network will try to learn these errors as well (in effect learning the way the algorithm works, not a general pitch estimation protocol).

What is needed is an unsupervised learning method for DRNN. The errors produced by the pitch estimator may be corrected later on, but what if the errors are severe or there is no current target available? Temporal clustering can be a strong feature of DRNN's, but as long as no real unsupervised learning methods are available, only a priori known input/output clustering can be learned.

5.6.1 Computational problems with large networks

A network of 200 neurons requires calculation of about $(5N)^2$ different equations *per time step*.

The learning phase adds another $(2N)^2$ calculations. For example, a network of 200 neurons requires for the pitch estimation learning process (550 datapoints per example) about $(5 \cdot 200)^2 \cdot 550 + 200^2 = 550.16 \cdot 10^6$ calculations. If the training requires approximately 60,000 learning epochs, the total number of calculations is $3.3 \cdot 10^{13}$. Note that each equation has in itself several simple computations like adding and multiplication. With some output to file, the current implementation takes six days to complete these calculations (with only sixty neurons). For larger networks, the computational burden becomes unpractical. There are some solutions to this problem:

- Parallel implementation of the algorithms
- Try to segment the given task into smaller problems, thus requiring smaller networks

5.7 Future research

The results described in section 5.5 are preliminary. Much is yet unknown and interesting issues are still open. Some of these are:

- **Noise.** How does a trained network react to signals with noise? Will it perform almost as good as it does with clean data? If so, it would truly outperform the current estimator
- **Training with all segments.** The current algorithm uses only the lower 63 of the 140 cochlea segments to give an estimate. Researchers claim that the upper part of the cochlea does not have enough resolution since these harmonics are too close apart or masked by other information. But maybe the network is able to detect these subharmonics and use them to give better estimates
- **On-line learning.** Check the detected pitch and correct the network if it is a 'near miss': if the estimate is a little off, the network could be correct. In this way the network sees an enormous amount of data, which it can hopefully use to generalize even better

This chapter indicates that the DRNN's have much potential, but further research is required in all different sorts of scientific and practical fields. But from the results in this chapter, it is clear a DRNN performs especially well in temporal and spatial clustering.

6 A derivation of learning rules for dynamic recurrent neural networks

A chapter by Henk A.K. Mastebroek[6] (h.a.k.mastebroek@bcn.rug.nl) of the department of Neurobiophysics and Biomedical Engineering. Copied with permission. This chapter brings together mathematical and physical tools which lie at the root of a derivation of learning rules for dynamic recurrent neural networks. It starts with the basic problem in the calculus of variations: how to determine a set of functions $y_i(x), i = 1, \ldots, n$***, which extremize the integral of some functional of these functions. This problem is generalized by the introduction of constraints such that the solutions also have to satisfy the constraint equations. This results in the method of the "Lagrange undetermined multiplier functions". Application of these methods in physics will be discussed after the introduction of Hamilton's Principle. The Lagrangian and Hamiltonian energy functions together with the canonical equations of motion of Hamilton are introduced. The way in which Pontryagin applied these methods in the construction of his "Minimum Principle of Pontryagin" for the solution of general optimal control problems is presented. The chapter ends with the derivation of the learning rules.***

6.1 A look into the calculus of variations

Let $y(x)$ be a function of x and let $f(y(x), y'(x);x)$ be a given functional of the independent variable x and the dependent variable $y(x)$ (with its derivative $y'(x) = \frac{d}{dx}y(x)$).The fundamental problem of the calculus of variations is now to determine the function $y(x)$ in such a way that the integral

Eq 6.1

$$J = \int_{x_1}^{x_2} f(y(x), y'(x);x)dx$$

is an extremum, i.e. a maximum or a minimum. x_1 and x_2 often will be fixed points, but they may vary. In order to find an extremum of J, the function $y(x)$ has to be varied. If a function $y(x)$ minimizes J, this means that all other "neighboring functions", even the closest ones, must make J increase. A neighboring funtion can be constructed as follows: let $y = y(\alpha, x)$ be a parametric representation in such a way that, if $\alpha = 0$: $y = y(0, x) = y(x)$, i.e. the function that minimized J. Now it is possible to write:

Eq 6.2

$$y(\alpha, x) = y(0, x) + \alpha\eta(x)$$

with $\eta(x_1) = \eta(x_2) = 0$, $\eta(x)$ being otherwise a function of x with continuous first derivative. The integral J now becomes a function of α too:

Eq 6.3

$$J(\alpha) = \int_{x_1}^{x_2} f(y(\alpha, x), y'(\alpha, x); x)dx$$

and for J to be an extremum it is necessary that:

Eq 6.4

$$\left.\frac{\partial J}{\partial \alpha}\right|_{\alpha = 0} = 0$$

for all functions $\eta(x)$. Euler showed in 1744 that the elaboration of this condition results in (here we refer to Thornton (1995), Kibble and Berkshire (1996)):

Eq 6.5

$$\frac{\partial f}{\partial y} - \frac{d}{dx}\frac{\partial f}{\partial y'} = 0$$

the well-known Euler equation.

Example I: Find the shortest path between the origin $O(x = 0, y = 0)$ and an arbitrary point $P(x = a, y = b)$ in the XOY-plane.

Solution: Let O and P be connected by a path according to the function $y = y(x)$. The length L of the connection is then: $L = \int_0^a \sqrt{(1 + y'(x)^2)}dx$.

So, in terms of the Euler equation the functional f is: $f = \sqrt{1 + y'(x)^2}$.

(Note that here f is independent of $y(x)$). From Eq 6.5 we now find: $\frac{d}{dx}\left[\frac{y'(x)}{1 + y'(x)^2}\right] = 0$.

The expression inside the brackets must be a constant, i.e.: $y'(x)$ has to be independent of x: $dy(x)/dx$ = constant, or: $y = cx + d$ (c and d being constants): i.e. a straight line with $c = b/a$ and $d = 0$ if it has to connect O and P.

It is not difficult to generalize the Euler equation to a set of equations for the case that f is a functional of, let us say, n independent variables:

Eq 6.6

$$f \equiv f(y_1(x), y_1'(x), \ldots, y_n(x), y_n'(x); x)$$

The development of the optimization problem now results in:

Eq 6.7

$$\frac{\partial f}{\partial y_i} - \frac{d}{dx}\frac{\partial f}{\partial y_i'} = 0$$

with $i = 1, 2, \ldots, n$, which we shall use below.

6.2 Conditions of constraint

In example I the problem was to find the shortest connection between two points in a plane. A more general problem is: find the shortest connection in 3-D between two points on a sphere (a "geodesic"). Now, there is an extra condition: the constraint that the solution of the minimization procedure (i.e. the shortest connection) has to satisfy the equation of the spherical surface. In the calculus of variations this problem is solved in an elegant way by means of introducing "Lagrange (undetermined) multiplier functions". We summarize the general case here (Thornton, 1995):

Let f be a functional as in Eq 6.6, and let $g_j(y_i(x); x) = 0$ ($j = 1, 2, \ldots, m$) be m equations of constraint. The solution of this type of extremum problem is given by the $(n + m)$ equations

Eq 6.8

$$\frac{\partial f}{\partial y_i} - \frac{d}{dx}\frac{\partial f}{\partial y_i'} + \sum_j \lambda_j(x)\frac{\partial g_j}{\partial y_i} = 0 \qquad i = 1, 2, \ldots, n$$

$$g_j(y_i; x) = 0 \qquad j = 1, 2, \ldots m$$

where the $\lambda_j(x)$ are the Lagrange multiplier functions. We have to solve now $(n + m)$ equations and will find $(n + m)$ solutions: n variables $y_i(x)$ and m functions $\lambda_j(x)$, where the solutions y_i satisfy the equations of constraint. Later on this solving technique will play an important role.

Example II (Note: a very simple problem that can be solved easier in other ways!): Maximize the area of a rectangle with perimeter P. Solution: Let the lengths of the sides be y_1 and y_2. The problem is then to maximize $f(y_1, y_2) = y_1 y_2$ under the constraint:

$g(y_1, y_2) = 2y_1 + 2y_2 - P = 0$ (note that f does not depend here explicitly on an independent variable, nor on the derivatives of the dependent variables). Eq's 6.8 become (λ too does not depend in this simple case on an independent variable): $y_2 + 2\lambda = 0, y_1 + 2\lambda = 0$ and: $2y_1 + 2y_2 = P$. It follows immediately that $\lambda = -P/8$, so that $y_1 = P/4$ and $y_2 = P/4$: the largest rectangle is a square.

6.3 Applications in physics: Lagrangian and Hamiltonian dynamics

From laboratory experience as well as daily life it follows that Newton's laws describe e.g. the behavior of a particle in an inertial frame in a correct way. Newton's second law $\bar{F} = \dot{\bar{p}}$ ($\bar{F}$: force, $\dot{\bar{p}}$: momentum) is a second order differential equation: $\bar{F} = \dot{\bar{p}} = m\dot{\bar{v}} = m\ddot{\bar{x}}$ (m: mass of the particle, $\dot{\bar{v}}$: velocity, $\ddot{\bar{x}}$: coordinate vector in the frame of the particle). It gives the solution for the path $\bar{x}(t)$ of the particle as soon as $\bar{F}$ is known. Note that the solution of the equation (of second order) needs two initial boundary conditions. In more complicated situations however, the equations of motion can become quite complex, and solutions may be very difficult to obtain. An alternative method of approach in the study of dynamical problems was proposed therefore by Hamilton in 1834/35 when he published the following dynamical principle ("Hamilton's Principle") on which all principles of mechanics and most of classical physics can be based (Thornton, 1995; Landau and Lifschitz, 1991):

"Of all the possible paths along which a dynamical system may move from one point to another within a specified time interval (consistent with any constraints), the actual path followed is that which minimizes the time integral of the difference between the kinetic and potential energies"

So, against the background of the calculus of variations, Hamilton's principle can be written as (we confine ourselves to conservative systems):

Eq 6.9
$$\delta \int_{t_1}^{t_2} (T - U)dt = 0$$

with $T = T(\dot{x}_i)$ the kinetic energy of the particle (a function of velocity $\dot{x}_i$ only) and $U = U(x_i)$ the potential energy (a function of coordinate x_i only). Define now a function

Eq 6.10
$$L \equiv T - U = L(x_i, \dot{x}_i)$$

Note that L is a functional like f in Eq 6.1. Substituting Eq 6.10 in Eq 6.1 and minimizing according to the Hamilton principle gives (in a rectangular coordinate system and for the single particle we are looking at):

Eq 6.11
$$\frac{\partial L}{\partial x_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{x}_i} = 0 \qquad i = 1, 2, 3$$

These are the well-known Lagrange equations of motion (similar to the Euler equations (4.7) with L the Lagrangian of the particle. Again these are second order equations (as in Newton's second law). In Eq 6.11 no forces appear, only energies!

Example III: Find the Lagrange equation of motion for the 1-D harmonic oscillator (mass m, spring constant k).

Solution: $L = T - U = (m\dot{x}^2)/2 - (kx^2)/2$. Therefore: $\frac{\partial L}{\partial x} = -kx$, $\frac{\partial L}{\partial \dot{x}} = m\dot{x}$ and $\frac{d}{dt}\frac{\partial L}{\partial \dot{x}} = m\ddot{x}$. So: $m\ddot{x} + kx = 0$, the same result of course as obtained in Newtonian mechanics.

6.4 Generalized coordinates

Let's consider a system of n particles, to be described in their motions with rectangular coordinates. We then need $3n$ independent coordinates. In the case that there are constraints on some of these coordinates (the motion could be constrained to occur on a certain surface), then, with m equations of constraint, $3n - m$ coordinates are independent: the system under consideration is said to have $s = 3n - m$ degrees of freedom. Generalized coordinates are now defined as a set of quantities that completely describe the system. They are denoted by $q_j (j = 1, 2, \ldots, 3n$ or $j = 1, 2, \ldots, s)$ and any s (or $3n$) independent quantities that completely describe the system will do. Sometimes it will be more convenient to choose generalized coordinates with dimensions of

energy, (length)^2 or even dimensionless quantities. In practice a set of generalized coordinates will be chosen (of course) that gives rise to equations of motion which are easy to handle. The time derivatives $\dot{q}_j$ of q_j are called generalized velocities. Different sets of generalized coordinates for one and the same dynamical system are interrelated via the so-called canonical transformations.

Example IV: The movement of the mass in the harmonic oscillator configuration of example III is completely determined by its coordinate $x(t)$ and its velocity $\dot{x}(t)$ at any time t, but also by the total energy E of the mass-spring system and the quotient of mass and spring constant m/k (Leech, 1958). Note that E and m/k are constants of this system.

It is not important whether we express the Lagrangian in terms of coordinates x_i and $\dot{x}_i$ or with the generalized coordinates q_j and $\dot{q}_j$: $L = L(q_j, \dot{q}_j; t)$; in this generalized notation Hamilton's principle is written now as:

Eq 6.12
$$\delta \int_{t_1}^{t_2} L(q_j, \dot{q}_j; t) = 0$$

which results in the generalized Lagrange equations of the system under consideration:

Eq 6.13
$$\frac{\partial L}{\partial q_j} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} = 0 \qquad j = 1, 2, \ldots, 3n$$

(Lagrange derived these equations along mathematical lines already in 1788, so without the use of Hamilton's principle).

An important result of the calculus of variations in relation to the use of generalized coordinates concerns the solution of the Lagrange equations of motion for constrained systems: when the constraint relations are in differential form, they can be included in the Lagrange equations by means of undetermined Lagrange multipliers, i.e. for n particles with m equations of constraint on their $3n$ coordinates we may write:

Eq 6.14
$$\frac{\partial L}{\partial q_j} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_j} + \sum_k \lambda_k(t)\frac{\partial f_k}{\partial q_j} = 0$$

with $j = 1, 2, \ldots, 3n$ and $k = 1, 2, \ldots, m$. (The multiplier functions $\lambda_k(t)$ have an important physical meaning which we do not discuss here (see e.g. Thornton (1995), Leech (1958)).

For a closed system (i.e. a system that does not interact with the world outside the system) in an inertial coordinate frame it can be proved that (Thornton,1995; Landau and Lifschitz,1991):

Eq 6.15
$$L - \sum_j \dot{q}_j \frac{\partial L}{\partial \dot{q}_j} = -H$$

and:

Eq 6.16
$$H = E = T + U$$

with H the "Hamiltonian" of the system, equal (under the restrictions mentioned before) to E, the total energy of the system. From the definition of $L = T - U$ it follows that the momentum component p_i of a particle along the i^{th}-axis is given by: $p_i = \frac{\partial L}{\partial \dot{x}_i}$. A generalized momentum component p_i can be defined now according to:

Eq 6.17
$$p_i \equiv \frac{\partial L}{\partial \dot{q}_i}$$

Note that this means that H can be written as $H = \sum_i p_i \dot{q}_i - L$. Straightforward reasoning (Thornton, 1995; Leech, 1958) results now in:

Eq 6.18
$$\dot{q}_k = \frac{\partial H}{\partial p_k} \quad \text{and:} \quad -\dot{p}_k = \frac{\partial H}{\partial q_k}$$

which are the famous "Hamilton's equations of motion", also called "canonical equations of motion". They constitute the foundation of "Hamiltonian dynamics". Note the symmetry in the role of the generalized coordinates p_k and q_k in these equations! p_k and q_k are called "canonically con-

jugate" or "adjoint" quantities. $\bar{p}$ is a covariant vector of the vector $\bar{q}$. Finally: $2s$ canonical equations of first order replace now s Lagrange equations of second order. In both cases we will need $2s$ initial conditions to be able to solve the equations. In many cases the canonical equations are easier to handle than the Lagrange equations, but this is not per se the case. The real power and charm of the Hamiltonian methods is to be found in the fact that they provide extensions from classical dynamics into many other fields of physics and beyond. The sections that follow are such extensions.

6.5 Application to optimal control systems

In the theory of control systems (Schultz and Melsa, 1967; Gamkrelidze, 1978) the state of a system at any time t_0 is defined as the minimum set of numbers $x_1(t), x_2(t), \ldots, x_n(t)$ which, together with the input to the system for $t \geq t_0$, are sufficient to determine the behaviour of the system for all $t \geq t_0$.

Generally, an n^{th} order linear control system is represented in state-variable form as a set of n first-order differential equations:

Eq 6.19
$$\dot{\bar{x}}(t) = A\bar{x}(t) + B\bar{u}(t)$$

with: $\bar{x}(t)$ the n-dimensional state vector, $\bar{u}(t)$ an r-dimensional control vector (input-like vector). A is the nxn system matrix and B the nxn control matrix.

In order to judge the performance of a system it is necessary to define a measure for it; this is done by introducing a so-called performance index (*PI*) or error function, as:

Eq 6.20
$$PI = \int_{t_1}^{t_2} L(\bar{x}, \bar{u}; t)dt$$

in such a way that the system is said to perform from t_1 to t_2 (the control period) in an optimal way when *PI* is minimized. Note that Eq 6.20 looks like Eq 6.1, except for the appearance of $\bar{u}$ instead of $\dot{\bar{x}}$ in Eq 6.20. L is chosen such that it is a positive definite function of $\bar{x}$, $\bar{u}$ and t, i.e. *PI* is monotonically increasing during the control period from t_1 to t_2.

The Pontryagin approach proceeds as follows: Write Eq 6.19 in short as:

Eq 6.21
$$\dot{\bar{x}}(t) = f(\bar{x}(t), \bar{u}(t); t)$$

(this is a non-linear time-varying equation, adequate for allmost all practical control problems!) and formulate the basic optimal control problem as the search for the optimal control $\bar{u}(t)$ which brings the system Eq 6.20 from state $\bar{x}^i(t^i)$ (*i* from *initial*) to state $\bar{x}^f(t^f)$ (*f* from *final*) while minimizing the errorfunction *PI* (Eq 6.20). The trajectory then described by the state vector $\bar{x}(t)$ is called the optimal trajectory. In a nutshell: the problem is to minimize the function in Eq 6.20 under the constraints of the system's equations as found in Eq 6.21.

In this case where the $\bar{x}^i$ and $\bar{x}^f$ values are given, the problem at hand is called the "basic optimal control problem". Pontryagin introduced here (Schulz and Melsa, 1967; Gamkrelidze, 1978), inspired by the technique of the undetermined Lagrange multipliers, a new state function $H(\bar{x}, \bar{u}, \bar{p}; t)$, the "state function of Pontryagin" (the "control Hamiltonian") with $\bar{p}$ the vector of multiplier functions (similar to what was called λ above):

Eq 6.22
$$H(\vec{\bar{x}}, \vec{\bar{u}}, \vec{\bar{p}}; t) = L(\vec{\bar{x}}, \vec{\bar{u}}; t) + \sum_{j=1}^{n} p_j(t) f_j(\vec{\bar{x}}, \vec{\bar{u}}; t)$$

Note that the last term in Eq 6.22 -in vector form- can be written as: $\bar{p}^T \bar{f}^T(\bar{x}, \bar{u}; t)$ ("*T*" from transpose form). Pontryagin showed that straightforward calculation and application of the calculus of variations now brings the following results (Schulz and Melsa, 1967):

Eq 6.23
$$\dot{\bar{x}} = \frac{\partial}{\partial \bar{p}} H(\bar{x}, \bar{u}, \bar{p}; t)$$

Eq 6.24
$$\dot{\bar{p}} = -\frac{\partial}{\partial \bar{x}} (H(\bar{x}, \bar{u}, \bar{p}; t))$$

Eq 6.25
$$O = \frac{\partial}{\partial \bar{u}} H(\bar{x}, \bar{u}, \bar{p}; t)$$

In other words: the problem of finding the optimal control $\bar{u}^0 = \bar{u}^0(\bar{x}, \bar{p}; t)$ is solved now by solving Eq 6.23-Eq 6.25.

First solve Eq 6.25 in order to find the $\bar{u}^0$ which minimizes H and substitute the result in the H function to obtain the "optimal Pontryagin state function":

Eq 6.26
$$H^0(\bar{x}, \bar{p}; t) = H(\bar{x}, \bar{u}^0(\bar{x}, \bar{p}; t), \bar{p}; t)$$

Eq 6.23 and Eq 6.24 now become:

Eq 6.27
$$\dot{\bar{x}} = \frac{\partial}{\partial \bar{p}} H^0(\bar{x}, \bar{p}; t)$$

Eq 6.28
$$\dot{\bar{p}} = -\frac{\partial}{\partial \bar{x}} (H(\bar{x}, \bar{p}; t))$$

which constitute a set of $2n$ coupled first order differential equations. We need the $2n$ boundary conditions $\bar{x}^i$ and $\bar{x}^f$ for their solution. This brings about a problem: it is not possible to integrate Eq 6.27 and Eq 6.28 both forward in time from initial conditions for both sets of equations, nor backward in time from final boundary conditions for both sets. So, a suitable computational scheme has to be constructed here.

The reader will have seen the strong resemblance of the developed architecture so far, with the Hamiltonian dynamics approach. There is a change in sign for the L function from Eq 6.20 in its use in the definition of the state function of Pontryagin in Eq 6.22. The $\bar{p}$ vector is "adjoint" to the $\bar{x}$ vector here too, but created solely now for reasons of mathematical conveniance: the $\bar{x}$ vector alone describes the system completely. So far the so-called "basic optimal control problem".

The "general optimal control problem" differs from the basic one in that -as initial conditions- the state values $\bar{x}^i(t^i)$ are given, but that the terminal boundary conditions $\bar{x}^f(t^f)$ can be fixed, free or may be specified by other conditions. We follow the Pontryagin approach to this problem in order to get insight in the construction of the tools we need in the design of appropriate learning rules for the recurrent neural networks. The solution begins with a generalisation of the energy function PI by adding a "terminal condition error function" $S(\bar{x}^f(t^f); t^f)$ according to:

Eq 6.29
$$PI = \int_{t^i}^{t^f} L(\bar{x}, \bar{u}; t)dt + S(\bar{x}^f(t^f); t^f)$$

The choice of S depends on the problem to be solved, S can be set to zero. Straightforward application of the calculus of variation methods (Schulz and Melsa, 1967; Gamkrelidze, 1978) delivers that the minimization of Eq 6.29 under the constraint equations Eq 6.21 results in the equations Eq 6.25 (with as a result Eq 6.26), Eq 6.27 and Eq 6.28. Now, these latter two equations can be solved only after the specification of the additional terminal boundary condition function $S(\bar{x}, t)$. In the case that $\bar{x}(t^f)$ and t^f are free, it turns out (Schultz and Melsa, 1967) that the generalized boundary condition becomes:

Eq 6.30
$$\left[\frac{\partial}{\partial \bar{x}} S(\bar{x}, t) - \bar{p}\right]^T \cdot d\bar{x}\Big|_{t^f} + \left[H^0(\bar{x}, \bar{p}; t) + \frac{\partial}{\partial t} S(\bar{x}, t)\right] \cdot dt\Big|_{t^f} = 0$$

delivering the extra conditions which are necessary for the solution of Eq 6.27 and Eq 6.28, i.e. for the minimization of Eq 6.29. Note that, if $S(\bar{x}, t)$ is taken to be zero, the boundary conditions for $\bar{p}(t^f)$ according to Eq 6.30 are: $\bar{p}(t^f) = 0$.

A rigorous proof of this "Minimum Principle of Pontryagin" can be found in e.g. Pontryagin et al. (1962), Boltyanski et al. (1966) and Fan (1966).

6.6 Time-dependent recurrent backpropagation: learning rules

Let us concentrate on a recurrent temporally continuous neural network with N neurons and $\mathcal{O}$ the subset of output neurons. The network equations are:

Eq 6.31
$$T_i \frac{d}{dt} y_i(t) = -y_i(t) + F(x_i(t)) + \theta_i \quad \text{with} \quad x_i = \sum_k w_{ki} y_k$$

and $i = 1, \ldots N$; $y_i(t)$ is the activation level of neuron i (note that $\bar{y}$ takes the place of $\bar{x}$ in section 6.5), T_i its time constant, $F(x_i(t))$ the activation function, x_i the total input of the neuron, the w_{ki} the weights for the k incoming signals y_k on neuron i and θ_i an external input or bias. The time constants will be considered in the learning process too because they play an important role in the dynamical performance of the network (Draye et al., 1995; 1996).

Suppose we want the network output neurons to generate an output (or target) signal $d_i(t)$ ($i \in \mathcal{O}$) (Pearlmutter, 1989). We define as a cost function:

Eq 6.32
$$q(\bar{y}(t), t) = \sum_{i \in \mathcal{O}} \frac{(d_i(t) - y_i(t))^2}{2}$$

and as an error (or "energy") function:

Eq 6.33
$$E = \int_{t^i}^{t^f} q(\bar{y}(t), t) dt$$

with $[t^i, t^f]$ the time interval during which the error function is calculated from the cost function. The reader will recognize in Eq 6.32 the equivalent of the L function and in Eq 6.33 the analogon of the performance index function PI as defined in section 6.5.

We want to calculate $\partial E / \partial w_{ij}$ and $\partial E / \partial T_i$ in order to optimize the weight factors and time constants. The state function of Pontryagin which we need for the calculation of the gradients of E becomes, now according to Eq 6.21 and Eq 6.22, taking $\theta_i = 0$:

Eq 6.34
$$H(\bar{y}, \bar{p}; t) = q(\bar{y}(t), t) + \sum_j p_j(t) \frac{d}{dt} y_j(t) = q(\bar{y}(t), t) + \sum_j \frac{1}{T_j} p_j(t)(-y_j(t) + F(x_j(t)))$$

where $\bar{p}(t)$ is the vector adjoint to $\bar{y}(t)$. Note that the "control vector" $\bar{u}$ from Eq 6.22 is not an input vector now, but is "hidden" in the weights w_{ij} and the time constants T_i.

If this H function would be optimal, and if we had $2N$ boundary conditions, we could solve for $\bar{y}$ and $\bar{p}$ via Eq 6.27 and Eq 6.28 and calculate the gradients of E, taking as a starting point that, at the optimum:

Eq 6.35
$$p_i(t) = \frac{\partial^+ E}{\partial y_i(t)}$$

where the + sign indicates that the "ordered derivative" is meant which gives $p_i(t)$ as a measure of how a small change in $y_i(t)$ at time t influences E including direct as well as indirect effects when this change propagates with time (Pearlmutter, 1995; Werbos, 1988).

However, H is not optimal and we have no $2N$ boundary conditions. Therefore we choose the "terminal condition error function" S (Eq 6.29) to be zero. This means that, according to Eq 6.30, $\bar{p}(t^f) = 0$ because $d\bar{y}(t^f)$ is not known and, due to the fact that t^f is fixed, $dt^f = 0$.

This brings an extra N boundary conditions! Further we suppose H to be the optimal H^0. Then, according to Eq 6.28 (where $\bar{y}$ takes the place of $\bar{x}$ now) and Eq 6.34:

Eq 6.36
$$\frac{dp_i}{dt} = -\frac{\partial q}{\partial y_i} + \frac{1}{T_i^0} p_i - \sum_j \frac{1}{T_j} w_{ij}^0 F'(x_j) p_j$$

with w_{ij}^0 the optimal weights and T_i^0 the optimal time constants (constituting together the optimal control).

Eq 6.31 (with our choice for $\theta_i = 0$) can be solved now forward in time, given the initial condi-

tions we set for the $y_i(t)$ at t^i, whereas Eq 6.36 can be solved backward in time, given the final conditions $p_i(t^f) = 0$. This results in the solutions for $y_i(t)$ and $p_i(t)$, which can be used now in the calculation of the gradients of E.

In doing so, we rewrite in an approximation Eq 6.31 as:

Eq 6.37
$$y_i(t+1) = \left(1 - \frac{\Delta t}{T_i}\right) y_i(t) + \frac{1}{T_i} F(x_i(t))\Delta t$$

where we took θ_i to be zero again. This relation enables us to calculate how much a variation of e.g. $y_i(t)$ or w_{ij} affects $y_i(t+\Delta t)$. Let w_{ij} change with an infinitesimal change dw_{ij} during a short period Δt beginning at t^i. Due to this change in w_{ij}, x_j changes with $dw_{ij}y_i(t)$ and, as a consequence, the second term in the right hand side of Eq 6.37 changes with $(1/T_j)dw_{ij}y_i(t)F'(x_j(t))\Delta t$ which is the variation of $y_i(t+\Delta t)$ too. According to Eq 6.35 the change in E now is: $(1/T_j)dw_{ij}y_i(t)F'(x_j(t))\Delta t p_j(t)$, and because we want to calculate the total effect of the change due to dw_{ij} on E over the whole time interval $[t^i, t^f]$, we find via integration:

Eq 6.38
$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_j}\int_{t^i}^{t^f} y_i(t)F'(x_j(t))p_j(t)dt$$

In the same way we find for the gradient:

Eq 6.39
$$\frac{\partial E}{\partial T_i} = -\frac{1}{T_i}\int_{t^i}^{t^f} p_i(t)\frac{d}{dt}y_i(t)dt$$

These equations provide us together with the $y_i(t)$ and $p_i(t)$ from Eq 6.31 and Eq 6.36 the tools we need to minimize E as a function of the "control", i.e. the weights w_{ij} and the time constants T_i by means of gradient descent.

For other derivations (along other lines of reasoning and using different methods) of learning rules for dynamic recurrent networks, we refer to Pearlmutter (1995).

6.7 References

Boltyanskii, V.G., Gamkrelidze, R.V., Mishchenko, E.F. and Pontryagin, L. S. (1966): The maximum principle in the theory of optimal processes of control. In: Optimal and self-organizing control, R. Oldenburger, Ed. Cambridge, MA: MIT Press

Draye, J-Ph.S., Pavisic, D.A., Cheron, G.A. and Libert, G.A. (1995): Adaptive time constants improve the prediction capability of recurrent neural networks. Neural Processing Letters 2(3), 12-16

Draye, J-Ph.S., Pavisic, D.A., Cheron, G.A. and Libert, G.A. (1996): Dynamic recurrent neural networks: A dynamical analysis. IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics 26, 692-706

Fan, L.T. (1966) : The continuous maximum principle. A study of complex systems optimization. J Wiley and Sons

Gamkrelidze, R.V. (1978): Principles of optimal control theory. Plenum Press.

Kibble, T.W.B. and Berkshire, F.H. (1996): Classical mechanics. Addison Wesley Longman Ltd

Landau, L.D. and Lifschitz, E.M. (1991): Mechanics. (Vol. I of the Course of Theoretical Physics). Pergamon Press

Leech, J.W. (1958): Classical mechanics. Methuen's monographs on physical subjects. Methuen and Co Ltd

Pearlmutter, B.A. (1989): Learning state space trajectories in recurrent neural networks. Neural Computation 1, 263-269

Pearlmutter, B.A. (1995): Gradient calculations for dynamic recurrent neural networks: A survey. IEEE Transactions on Neural Networks 6, 1212-1228

Pontryagin, L.S., Boltyanskii, V.G., Gamkrelidze, R.V. and Mishchenko, E. F. (1962): The mathematical theory of optimal processes. J Wiley and Sons

Schultz, D.G. and Melsa, J.L. (1967): State functions and linear control systems. McGraw-Hill

Thornton, M. (1995): Classical dynamics of particles and systems. Saunders College Publishing. Hartcourt Brace College Publishers

Werbos, P.J. (1988): Generalization of backpropagation with application to a recurrent gas market model. Neural Networks 1, 339-356

7 References

A list containing all the used references in this document

[1] *V.G. Boltyanskii, R.V. Gamkrelidze, E.F. Mishchenko and L.S. Pontryagin;* 1996; **The Maximum Principle in the theory of optimal processes of control**; Optimal and Self-orginazing Control, pages 262-266

[2] *Jean-Philippe Draye;* 1996; **Dynamic Recurrent Neural Networks for Adaptive Temporal Processing**; dissertation

[3] *Simon Haykin*;1994; **Neural Networks, A comprehensive foundation**; Prentice Hall International Edition; ISBN 0-13-895863-7

[4] *Dik. J. Hermes*; januari 1988; **Measurement of pitch by subharmonic summation**; Journal for the Acoustical Society of America

[5] *James W. Kalat;* 1993; **Biological Psychology**; Brooks/Cole Publishing Company; ISBN 0-534-21108-9

[6] *H.A.K. Mastebroek and J.E. Vos (editors)*; 2001; **Plausible neural networks for biological modelling**; Kluwer Academic Press; ISBN -

[7] *Tom M. Mitchell*; 1997; **Machine Learning**; McGraw-Hill; ISBN 0-07-115467-1

[8] *Dan W. Patterson;* 1996; **Artificial Neural Networks**; Prentice Hall; ISBN 0-13-295353-6

[9] *W.H. Press et.al.*; **Numerical Recipes in C**; second edition; Cambridge University Press; ISBN 0-521-43108-05