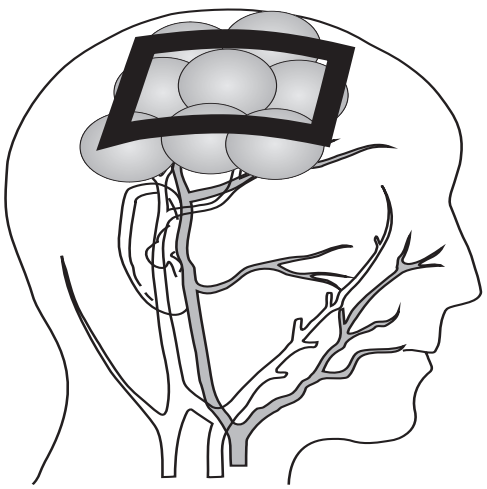
TEL - AVIV UNIVERSITY
Faculty of Engineering
Department of Electronics
Physical Electronics Program

Training Synaptic Delays in a Recurrent Neural Network.

Thesis submitted towards the degree of "Master of Science"

by

Barak Cohen



August 1994

TEL - AVIV UNIVERSITY
Faculty of Engineering
Department of Electronics
Physical Electronics Program

Training Synaptic Delays in a Recurrent Neural Network.

Thesis submitted towards the degree of "Master of Science"

by

Barak Cohen

This research work was carried out at Tel-Aviv University,
Faculty of Engineering,
Department of Physical Electronics
under the supervision of **Prof. E. Marom** and **Dr. D. Saad.**

August 1994

I would like to thank Dr. David Saad and Prof. E. Marom for their support, encouragement and inspiration throughout this research and to the Internet that encourages the science research around the globe.

Many Thanks to my family.

And to Daniella.

Abstract

Algorithms for training recurrent neural networks (RNN), both with and without time delays, to perform certain tasks are notorious as a major computer resources consumers. These algorithms adapt a set of network parameters for performing certain tasks typical for the network's architecture. The addition of the possibility for synaptic delays modification enlarges the parameter space of the network, thus making the training task even more demanding and the network more sensitive to the choice of parameters. The main incentives for adding this possibility are rooted in our knowledge about biological nets as well as in practical and applicable reasons, ranging from enhancing generalization to enhancing network capabilities. In this work we show how addition of time delay parameters results in incrementing both network capacity and capabilities and enabling the use of smaller and more competent nets for the same tasks. In addition, the current work introduces several methods to accomplish recurrent neural networks training, optimizing both the synaptic time delays and the weights. Several training methods are evaluated. The most efficient one, based on adaptive simulated an-

nealing, enables training of recurrent neural networks with synaptic time delays even on a low-end (Personal Computer) platform. The performance of the various training methods and the capabilities of recurrent neural networks with time delays as a whole are examined via extensive computer simulations using typical benchmark tasks. In addition, special new configurations of recurrent neural networks with time delays, modifications of the general configuration, are presented and examined together with special training algorithms.

Contents

1 Introduction **1**

1.1 Basic neural network configurations 1

1.1.1 Artificial neural networks - basic concept . . . 1

1.1.2 Recurrent neural networks 3

1.2 Training 5

1.3 Time delayed recurrent neural networks (TDRNN) . 9

1.4 Network dynamics 12

1.5 Performance measures 13

2 Training algorithms **15**

2.1 Gradient descent type algorithms 15

2.1.1 Direct parameter derivation 16

2.1.2 Indirect parameter influence 18

2.2 Teacher forcing 26

2.3 Learning parameters 27

2.4 Simulated annealing based training algorithms 29

2.4.1 Boltzmann Annealing (BA) 31

2.4.2 Cauchy Fast Annealing (CFA) 34

2.4.3 Adaptive Simulated Annealing (ASA) 36

2.4.4 Summary 40

3 Implementation **44**

3.1 Task description 44

3.2 Computer resources 47

3.2.1 Memory allocation 47

3.2.2 Time consumption 48

3.3 Execution overview 49

3.4 Network behavior 55

4 Other TDRNN configurations **61**

4.1 Direct gradient descent for MODEL 1 64

4.2 Direct gradient descent for MODEL 2 65

4.3 Indirect gradient descent for MODEL 1 65

4.4 Indirect gradient descent for MODEL 2 66

4.5 Implementations 67

4.6 Conclusions 69

5 Summary **72**

A Numerical integration methods **79**

B RNN implementations **83**

B.1 Real Time Recurrent Learning (RTRL) 84

B.2 State space trajectories in RNN 90

B.3 Adjoint Operators method 92

B.4 Signal reconstruction with TDFF 94

C Indirect gradient descent algorithm for TDRNN MODEL 2 **97**

List of Symbols and Abbreviations

NN	—	Neural Network
FF	—	Feed Forward (type of NN)
RNN	—	Recurrent NN
TDRNN	—	Time delayed RNN
GD	—	Gradient Decent
PC	—	Personal Computer
BP	—	Back Propagation
SA	—	Simulated Annealing
ASA	—	Adaptive Simulated Annealing
VFSR	—	Very Fast Simulated Annealing
τ_{ij}	—	Synaptic delay of neural transmission from neuron i to j
w_{ij}	—	Synaptic connection strength from neuron i to j
T_i	—	Synaptic response factor of neurpon i
$\sigma()$	—	Neural function (activation)

List of Tables

2.1 A comparison between different simulated annealing techniques parameters: The probability g to choose next state, the energy threshold probability h and the cooling schedule that ensures convergence $T(k)$. . . . 40

3.1 Memory consumption of different training algorithms. The memory is stated in terms of the network size N, the trajectory length L and the time delay deepness S. 47

3.2 Time consumption of different training algorithms per iteration. Expressed in terms of the network size N, then trajectory length L and the time delay depth S. 48

3.3 Network size comparison. One notices that TDRNN requires significantly less neurons and connections. . . 50

3.4 Convergence time benchmark. One notices that the ASA algorithm converges time is of the same scale as other algorithms implementing regular RNN while gradient decent algorithm for training TDRNN require longer training procedure and might fail training. 51

3.5 ASA training performance summary. The ASA algorithm was proven as a stable algorithm for solving various tasks. 55

4.1 ASA training performance summary for different TDRNN models. Note that the performance of the basic model is slightly different from what was presented in chapter 3. This is due to different initial parameters that were chosen in order to make equal tests for the three models. 68

B.1 The performance of RTRL algorithm with two typical tasks as presented by Williams and Zipser. 85

B.2 The performance of indirect GD scheme for RNN . . 92

B.3 The performance of adjoint GD scheme for RNN . . . 92

List of Figures

1.1 The modeling of the biological neuron into an artificial form. 2

1.2 Types of neural networks , FF (left) , RNN (middle) and TDRNN (right). The gray arrows represent connections without time delays and the black arrows represent connections with time delays. 4

1.3 A homogenous TDRNN topology. Each circle represents a neuron with a nonlinear function σ operating on the neuron's activation for production of output signal . Each connection, represented by lines encapsulates both weight and time delay. 10

2.1 A two dimensional representation of BA probability g as a function of normalized parameter deviation and cooling temperature. 33

2.2 A two dimensional representation of FA probability g as a function of normalized parameter deviation and cooling temperature. 35

2.3 A two dimensional representation of ASA probability g as a function of normalized parameter deviation and cooling temperature. 37

2.4 A comparison between cooling schedules of three simulated annealing algorithms: The regular Boltzmann Annealing (BA) (top), the Cauchy Fast Annealing (CFA or FA) (middle) and the Adaptive Simulated Annealing (ASA) (bottom). One can notice how a faster schedule is achieved while selecting a suitable generating probability g. 42

2.5 A comparison between probability functions $g(x)$ of three simulated annealing algorithms: The regular Boltzmann Annealing (BA), the Caushy Fast Annealing (CFA or FA) and the Adaptive Simulated Annealing (ASA). Note that an annealing function with thicker "wings" enable faster training procedure as shown in figure 2.4. The graphs also show the change in the probability distribution as temperature decreases, in this case between 0.2 and 3. 43

3.1 Some of the tasks teacher signals. The circle, the eight pattern, the square and the justice pattern. The patterns are shown as X-Y plots where X values are output of one neuron and Y are output of the other. . 46

3.2 Convergence behavior of two types of algorithms. The left graph represents a typical behavior of the ASA algorithm. The right graph represents the behavior of gradient decent algorithm. One notices several irregularities in the GD graph, resulting from the rescaling procedure in the algorithm designed to tune the various parameters according to their sensitivities. . . . 49

3.3 The TDRNN output signals: The top graphs show the circle pattern as a function of time (left) and as an X-Y plot (right). The middle graphs show the quadrangle pattern as a function of time (left) and as an X-Y plot (right). The bottom graphs show the eight shape (left) and the BST (right). 53

3.4 The influence of local feedback with time delay on oscillation, the graphs show different solutions of $\frac{dy}{dt} = -y - w\sigma(y(t-\tau)) + h(0)$ for $\tau = 0.28, \tau = 0.58, \tau = 0.88$ and $w = 100$. One notices the changes in local oscillation frequency and amplitude. 56

3.5 A neural network that was trained to produce a "Quadrangle" trajectory with an input $I(t) = h(0)$ response for input perturbations. The original trajectory is shown in Figure 3.3. The graphs (from top to bottom) show the response of the TDRNN for the following perturbations in the input signal: $I(t) = h(0) + 0.5\sin(t)$ and $I(t) = h(0) - h(0.01)$, $I(t) = h(0) + 0.5 * \sin(t)$ and $I(t) = h(0) - \sin(10t)$. The right graphs show the trajectory on an X-Y plot while the left show the response as a function of time. On the left graphs one notices the changes in the behavior of the free neuron while after attaining the oscillation the other neurons maintain a stable trajectory. 60

4.1 A comparison between the structure of three different models of TDRNN: The basic model (left), MODEL1 (middle) and MODEL2 (right). 63

4.2 A comparison between the training schedule of three different models of TDRNN using ASA: the basic model (solid line), MODEL 1 (dot dashed) and MODEL 2 (dashed line) 71

B.1 RTRL training of a stable oscillation. One notices the coinciding of the trained oscillating signal over the desired signal. 86

B.2 RTRL training of two stable oscillations. The signals are drawn over the desired signals. The double dashed line at the bottom is the difference between the actual and desired signals. 87

B.3 An X-Y plot of the circle task as was learned by an RTRL algorithm. 88

B.4 An X-Y plot of the eight shape task as was learned by an RTRL algorithm. 89

Chapter 1

Introduction

1.1 Basic neural network configurations

1.1.1 Artificial neural networks - basic concept

Artificial neural networks are based on parallel identical simple processors simulating the main characteristic behavior of clusters of neurons in the brain. Each component (neuron) in the system implements a nonlinear transfer function and receives signals (from outside or from other components) through connections. These connections are the artificial versions of the biological synapses. Each connection withholds certain physical properties also called connec-

tion parameters. Unlike the biological models that encompass several complex components, the artificial version contains only few of the biological synapse properties. The basic artificial networks use

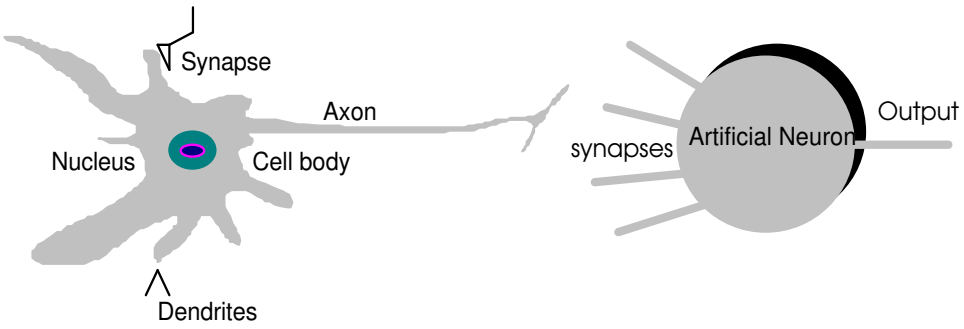


Figure 1.1: The modeling of the biological neuron into an artificial form.

only one parameter to model each connection, namely, the weight parameter that multiplies the signal in the specified connection. The input signals are fed into a non linear component that stands for the biological neuron cell. The artificial neuron is an abstraction of the biological cell and is generally represented by a non linear function (such as a sign function or sigmoid). The network itself is built out of several neurons and is characterized by the connections between them and the properties of the connections and of the neurons. The basic artificial networks consists of layers of neurons connected in a

unidirectional manner, which is also called Feed Forward (FF), since data flows forward from one layer to another. Although this representation is very simple, it is powerful enough to performing many tasks and has several practical implementations such as associative memory application, control problems and optimization.

1.1.2 Recurrent neural networks

Recurrent Neural Networks (RNN) are networks that include feed back connections in addition to FF connections used in most of the currently used Neural Networks (NN).

In FF neural networks the topology of the network determines that the neural activity will propagate through the inter neural connections (synapses) from the input ports in an unidirectional way to the output ports of the network. RNN configuration enables the neural activity to flow in any direction between the neurons. Figure 1 shows graphically the difference between the configurations of RNN and FF neural networks. It is obvious that FF networks are a specific case of RNN where all backward connections are discarded. Figure 1.2 compares graphically the structure of FF neural networks

versus RNN. One can classify RNN according to the following characteristics:

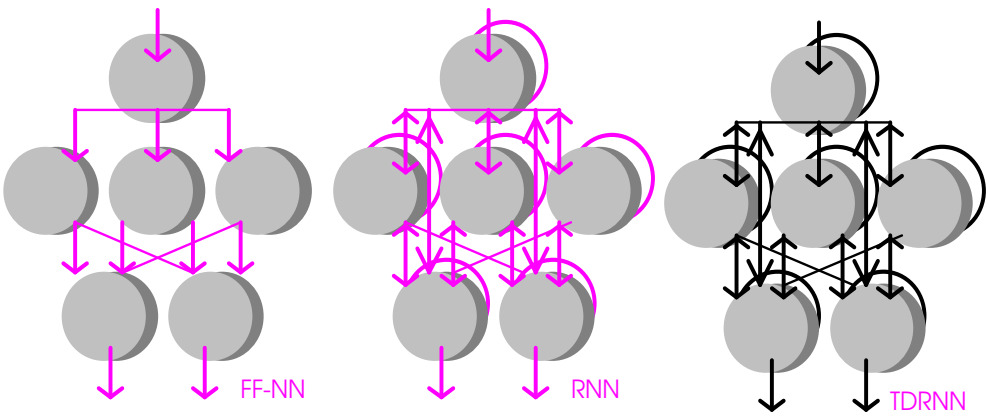


Figure 1.2: Types of neural networks , FF (left) , RNN (middle) and TDRNN (right). The gray arrows represent connections without time delays and the black arrows represent connections with time delays.

- *Activation:* in a homogenous RNN all neurons share a similar activation function. Non homogenous (heterogeneous) networks include several types of neurons and might have other linear or nonlinear components within its structure. In many cases non homogenous networks can be designed for special tasks, where a special component can make the task imple-

mentation easier [10].

- *Dynamics:* Neural dynamics can be discrete or continuous. In the case of continuous dynamics, the neurons produce continuos output and are interacting continuously. Discrete dynamics limits the neural network to work in a discrete time step propagation. It is important to note that due to computational limitation, we simulate either of the dynamics by discrete time steps. NN with discrete dynamics can propagate either synchronously, where the neural activity changes simultaneously, or asynchronously where only one neuron state is updated each time.

This work will concentrate on homogenous neural networks with continuos dynamics.

1.2 Training

The goal of any neural network is to perform a task (or several tasks). In order to do so the network should have certain properties: (i) a configuration, appropriate for performing the task (number and type

of neurons in the network and the type and qualities of the connections); and (ii) the value of the parameters that define the relations between the neurons. One way to set these properties is deterministic, one can pre-determine the properties of the network. Another way is to let these properties evolve in response to an external demand in a process called:*training*. Using a training procedure, we can determine the parameters and properties required for a certain NN to perform a certain task. This procedure is usually done with predetermined configuration of the network. Several works are dealing with the issue of how to determine what configurations can do which tasks and with self-configuring networks [22]. We will not discuss this matter in this work.

The currently used training algorithms for RNN handle various configurations, activation functions and dynamics. One can classify the different approaches to training RNN using the following categories:

- *Casual/Non casual/Real time training algorithms:* Training a NN can be accomplished according to several paradigms: One

is to let the system run freely and propagate forward in time while storing parameters and using them later on for implementing the learning procedure. This process is called casual training because both the system itself and the training process propagate forwards in time.

A non casual training algorithm involves a backward propagation through time during the training phase. This process can be implemented mathematically but is non casual since it requires the use of "future" parameters in a differential equation going backwards in time, used for modifying network parameters.

A real time training algorithms is a special case of casual algorithms where the parameter modification is carried out during the evolution of the network. In several works, where non casual training algorithms where implemented, an effort was made to transform non casual algorithms to casual [8].

- *Local/Non-local training algorithms:* A Local training algorithm is an algorithm where parameter modification of a neuron can be implemented by getting data from the adjacent connec-

tions only. The main advantage of local algorithms is that they fit parallel processing techniques. Non local algorithms use the overall data of the network for training.

The utilization of these methods is done by searching for the optimal parameters within a space of parameters. There are two main approaches to perform the search:

- *Deterministic methods (usually direct/indirect gradient descent methods):* These methods utilize deterministic approaches to optimization problems. Most of these algorithms are variants of Gradient Descent methods (GD). The direct GD approach uses derivation of a cost function with respect to the network parameters for estimating parameter influence on the network. Indirect GD approach uses related functions as an intermediate step to compute similar properties. The motivation for using the indirect approach is rooted in the effort to save computation resources (i.e. computation time and memory) by introducing more complex algorithms.

- *Stochastic methods:* Stochastic methods use statistical search

methods to overcome some problems that deterministic algorithms suffer from. These methods include genetic algorithms, simulated annealing, Boltzmann machine, etc. These methods are useful when the system is highly non linear and deterministic methods fail. These methods are proven to reach a global optimum for the parameters if the stochasticy of the algorithm is reduced slowly enough.

1.3 Time delayed recurrent neural networks (TDRNN)

Time delay recurrent neural networks (TDRNN), are an extension of the conventional recurrent neural network (RNN) allowing the use of synaptic time delays, varying from connection to connection. A TDRNN with synaptic delays of value 0, obviously coincide with the conventional RNN definition. The difference between TDRNN and RNN is that while in a conventional RNN, the information flows through the synapse instantly, (i.e., when one of the neurons changes its output value all other neurons are instantly affected), in TDRNN this information is delayed. Thus, information coming from neuron i will arrive to neuron j after a delay τ_{ij}. Figure 1.3 shows graphically

a typical TDRNN configuration. The incentives for advancing from RNN to TDRNN are:

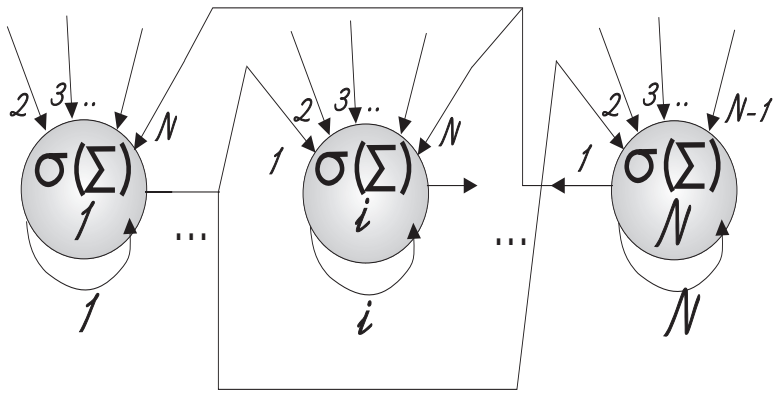


Figure 1.3: A homogenous TDRNN topology. Each circle represents a neuron with a nonlinear function σ operating on the neuron's activation for production of output signal . Each connection, represented by lines encapsulates both weight and time delay.

- **Increased capacity resulting from the extended configuration and added degrees of freedom** – These nets can handle tasks that cannot be performed by conventional RNN techniques such as generation of non-derivable functions.

- **Reducing network size** – results from the fact that one can perform similar tasks with a smaller number of free parameters due to the enhanced capabilities of the neural network and the larger number of degrees of freedom. Reducing the number of

relevant free parameters might also result in an enhancement of network generalization capabilities.

- **Transferring past information** – Information on past behavior of the system is usually stored as different internal representations requiring a larger number of hidden neurons but can also be provided using time delays with no additional hidden neurons.

- **Conforming with existing time delays in biological nets** – These synaptic time delays play an extremely important role in biological neural networks.

In fact some of these reasons provide incentives for progressing from feed-forward nets to RNN as well.

The drawback of the additional parameters is that the training procedure becomes significantly harder to implement. All RNN training algorithms suffer from a very long training procedure as well as sensitivity to the choice of parameters. Adding synaptic time delays makes the situation worse. In this work we will show the advantages of using TDRNN as well as present and examine

a couple of methods for training such nets effectively to perform several typical benchmark tasks..

1.4 Network dynamics

The objective of the TDRNN that were trained in this work is to produce a continuous trajectory both in its representation and its dynamics. The network dynamics used in this work was mentioned in ref. [5] and has been implemented until now without synaptic time delays. The neural network dynamics is described by the following equations:

$$\begin{aligned} x_i(t) &= \sum_{j=1}^{N} w_{ij} y_j(t - \tau_{ij}) \\ T_i \dot{y}_i(t) &= -y_i(t) + \sigma\left(x_i(t)\right) + I_i \end{aligned} \tag{1.1}$$

where $I_i(t)$ is an external dynamical input function to neuron i, $\sigma(x)$ is the transfer function defined by $\sigma(x) = \frac{1}{1+e^{-x}}$, $y_i(t)$ is the output of neuron i, T_i is a time constant related to each neuron controlling its dynamic behavior, w_{ij} is the weight matrix, x_i represents the total input to neuron i and N represents the number of neurons in

the system. The new parameter τ_{ij} represents the synaptic delay between neurons i and j.

1.5 Performance measures

A network of type (1.1) is designed and trained to perform a task, in our case, for a certain external activation the network should produce a predetermined output trajectory through time. The resemblance between the desired output trajectory and the actual one measures the success of the network in performing the task. The usual tool for providing this measure is a cost function of the form:

$$E(P) = \sum_{k=1}^{M} \left(\int_{t_0}^{t_1} (y_k(t) - f_k(t))^2 dt \right) \tag{1.2}$$

where P is a certain state of the network. $E(P)$ measurs the squared error between the actual trajectory $y_k(t)$ and the desired one $f_k(t)$. M represents the number of output neurons in the system. We will refer to this property also as the error property or the energy property later in this work. The final performance of the network (as mentioned above) is not the only performance measure one needs.

On the way to define the network configuration and parameters an adaptation procedure is used (training). The performance of this procedure is also an important factor that is measured by the computer resources that one uses. The resources are time and memory. An optimal training procedure will consume less time and less memory to accomplish the required final results.

This work will show that TDRNN performs better than RNN in both measures and will show and compare several training algorithms for TDRNN. Some of the algorithms are powerful enough to implement training on a low-end platform. In addition we will introduce other simple configurations of TDRNN and their training algorithms.

Chapter 2

Training algorithms

Several training algorithms for the TDRNN whose dynamics is defined in Eq. (1.1) and (1.2), are introduced in the following sections. In each section there is a thorough overview of each algorithm and it's development.

2.1 Gradient descent type algorithms

Gradient decent approach for solving optimization problems is common and is especially useful when dealing with known functions. Several versions of GD algorithms where implemented with RNN [7] and were capable of RNN to perform several simple tasks. In

this work two training algorithms based on gradient descent are examined:

2.1.1 Direct parameter derivation

This method [5, 7] is based on direct derivation of the energy function with respect to the parameters using an iterative procedure. This algorithm is non local [1] and causal and is commonly used for training RNN. The training algorithm is obtained by deriving Eq. (1.1) with respect to the various parameters. We first define the following terms:

$$\begin{aligned} p_{ij}^k &= \frac{\partial y_k(t)}{\partial w_{ij}} \\ q_i^k &= \frac{\partial y_k(t)}{\partial T_i} \\ r_{ij}^k &= \frac{\partial y_k(t)}{\partial \tau_{ij}} \end{aligned} \tag{2.1}$$

representing the sensitivity of the output to the various parameters. Deriving the dynamic equation (1.1) with respect to the different parameters w, T and τ respectively yields the following differential

[1] A non local training algorithm uses data from all neurons for training, contrary to local algorithms using information from adjacent neurons only

equations for the various terms in Eq. (2.2):

$$\begin{aligned} T_k \frac{dp_{ij}^k}{dt} &= -p_{ij}^k + \dot{\sigma}(x_k) \sum_l w_{lk} p_{ij}^l \\ T_k \frac{dq_i^k}{dt} &= -q_i^k + \dot{\sigma}(x_k) \sum_l w_{lk} q_i^l \\ T_k \frac{dr_{ij}^k}{dt} &= -r_{ij}^k + \dot{\sigma}(x_k) \left[w_{ij} \frac{dy_i}{dt}(t - \tau_{ij}) \delta_{jk} - \sum_l w_{lk} r_{ij}^l (t - \tau_{lk}) \right] \end{aligned} \tag{2.2}$$

where $\dot{\sigma}(x) = \frac{d\sigma}{dx}$. The δ function is defined in Eq. (2.3).

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

The parameters correction process is carried out by using the derivative of the energy with respect to the various parameters computed by propagating the above equations forward in time using $e_k(t) = (y_k(t) - f_k(t))^2$, as follows:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \int_{t_0}^{t_1} \sum_k e_k(t) p_{ij}^k(t) \\ \frac{\partial E}{\partial T_i} &= \int_{t_0}^{t_1} \sum_k e_k(t) q_i^k(t) \\ \frac{\partial E}{\partial \tau_{ij}} &= \int_{t_0}^{t_1} \sum_k e_k(t) r_{ij}^k(t) \end{aligned} \tag{2.4}$$

where the actual correction procedure for each parameter $P = [\mathbf{w}, \mathbf{T}, \tau]$ is obtained by using the sensitivity of the energy function to the parameters:

$$\frac{dP_l}{dt} = -C_l \frac{\partial E}{\partial P_l} \tag{2.5}$$

where C_l is a constant that controls the adaptation procedure of a certain parameter P_l.

2.1.2 Indirect parameter influence

This method [5, 8] is based on introducing additional variables, used as a tool for calculating the main relevant derivatives. The method is non local and non causal since it propagates the system forward and backwards through time generating a couple of functions, used for carrying out the gradient descent procedure. The sensitivity functions are defined by

$$z_i(t) = \frac{\partial E}{\partial y_i(t)}, \tag{2.6}$$

measuring the influence of an infinitesimal change of the trajectory at time t on the energy function through the effect of this change

on the rest of the trajectory (from t to the end of the trajectory). It is obvious that at the end of the trajectory ($t = t_f$) this function equals zero, therefore the evaluation of this function is obtained by backward propagation using the end conditions at t_f. To compute the function $z(t)$ we first introduce a discrete version of the dynamic equation (1.1) that leads to:

$$y_i(t + \Delta t) = \left(1 - \frac{\Delta t}{T_i}\right) y_i(t) + \frac{\Delta t}{T_i}\sigma\left(\sum_j w_{ji} y_j(t - \tau_{ji})\right) + \frac{\Delta t}{T_i} I_i(t). \tag{2.7}$$

In order to compute $z(t)$ we will change $y(t)$ by an infinitesimal value ϵ:

$$y_i(t+\Delta t, \epsilon) = \left(1 - \frac{\Delta t}{T_i}\right) (y_i(t)+\epsilon) + \frac{\Delta t}{T_i}\sigma\left(\sum_j w_{ji} y_j(t - \tau_{ji})\right) + \frac{\Delta t}{T_i} I_i(t) \tag{2.8}$$

Now we derive the equation by the chain rule:

$$\begin{aligned} z_i(t) = \frac{dE(t)}{d\epsilon} = & \underbrace{\sum_j \frac{\partial E}{\partial y_j(t + \Delta t)} \frac{\partial y_j(t + \Delta t)}{\partial \epsilon}}_{I} \\ & + \underbrace{\frac{dE}{d\epsilon}}_{II} + \underbrace{\sum_j \frac{\partial E(t + \Delta t)}{\partial y_j(t + \tau_{ij})} \frac{\partial y_j(t + \tau_{ij})}{\partial \epsilon}}_{III}, \end{aligned} \tag{2.9}$$

where term I is the influence through $y(t)$, term II is the direct influence and term III is the influence through $y(t+\tau_{ij})$. After introducing y into Equation (2.9), we get the three expressions:

$$\begin{aligned} I &= z_j(t+\Delta t)\left(1-\frac{\Delta t}{T_j}\right) \\ II &= e_i \Delta t \\ III &= \sum_j z_j(t+\tau_{ij})\frac{\Delta t}{T_i}\dot{\sigma}\left(\sum_l w_{li} y_l(t+\tau_{lj})\right) w_{ji} \end{aligned} \tag{2.10}$$

where $\dot{\sigma}(x) = \frac{d\sigma}{dx}$. The function z is therefore:

$$\begin{aligned} z_i(t) = e_i(t)\Delta t + z_i(t+\Delta t)(1+\frac{\Delta t}{T_i}) \\ + \sum_j z_j(t+\tau_{ij})\frac{\Delta t}{T_j} w_{ji}\dot{\sigma}\left(\sum_l w_{li} y_l(t+\tau_{li})\right) \end{aligned} \tag{2.11}$$

We can now compute the influence of the parameters on the energy using the function z. Similarly we compute the influence from the weights w_{ji} by examining the sensitivity of the error measure to an infinitesimal modification ϵ of w_{ji} at time t, defining a new function V_{ji}:

$$V_{ji} = \frac{dE}{d\epsilon} = \underbrace{\frac{\partial E}{\partial y(t+\Delta t)}}_{=z(t+\Delta t)} + \underbrace{\frac{\partial y(t+\Delta t)}{\partial \epsilon}}_{II} + \underbrace{\frac{dE}{d\epsilon}}_{=0} \tag{2.12}$$

and the sensitivity of the energy to the weights is therefore:

$$\frac{dE}{dw_{ji}} = \int_{\tilde{t}} \frac{dV_{ji}}{dt} dt \tag{2.13}$$

where $\tilde{t}$ represents the whole time span on which the functions are valid. The method for deriving the equation for V_{ji} is identical to the one used previously:

$$y_i(t+\Delta t, \epsilon) = \left(1 - \frac{\Delta t}{T_i}\right)(y_i(t)) + \frac{\Delta t}{T_i}\sigma\left(\sum_j (w_{ji} + \epsilon_{ji}) y_j(t - \tau_{ji})\right) + \frac{\Delta t}{T_i} I_i(t) \tag{2.14}$$

Term II from Eq. (2.12) becomes:

$$\frac{\partial y_i(t+\Delta t)}{\partial \epsilon} = \frac{\Delta t}{T_i}\dot{\sigma}\left(\sum_j w_{ji} y_j(t - \tau_{ji})\right) y_j(t - \tau_{ji}) \tag{2.15}$$

We obtain the following equation for V_{ji}:

$$\frac{\partial V_{ji}}{\partial t} = -\frac{z_i(t)}{T_i}\dot{\sigma}\left(\sum_j w_{ji}y_j(t-\tau_{ji})\right)y_j(t-\tau_{ji}) \tag{2.16}$$

The same modification of weight w_{ji} at time t, affects the output and energy at $t+\tau_{ji}$, thus, we shift the time $t \to t+\tau_{ji}$ to get:

$$\frac{\partial V_{ji}}{\partial t} = -\frac{z_i(t+\tau_{ji})}{T_i}\dot{\sigma}\left(\sum_j w_{ji}y_j(t)\right)y_j(t) \tag{2.17}$$

The energy sensitivity measure with respect to the weight modification is therefore:

$$\frac{\partial E}{\partial w_{ji}} = -\int_{t_0+\tau_{ji}}^{t_1+\tau_{ji}} \frac{z_i(t+\tau_{ji})\dot{\sigma}\left(\sum_j w_{ji}y_j(t)\right)y_j(t)}{T_i}dt \tag{2.18}$$

Similarly, for computing the influence of the coefficients T_i on the energy, we use function $\rho_i = \frac{1}{T_i}$. We then define a new function K_i, so that

$$\frac{dE}{d\rho_i} = \int_{t_0}^{t_f} \frac{dK_i}{dt}dt \tag{2.19}$$

We define therefore K_i as:

$$K_i = \frac{dE}{d\epsilon} = \underbrace{\frac{\partial E}{\partial y(t+\Delta t)}}_{=z(t+\Delta t)} + \underbrace{\frac{\partial y(t+\Delta t)}{\partial \epsilon}}_{II} + \underbrace{\frac{dE}{d\epsilon}}_{=0} \tag{2.20}$$

In order to get term II above, the discrete function y is defined with an infinitesimal change ϵ in the parameter ρ:

$$y_i(t+\Delta t, \epsilon) = (1 - \frac{\Delta t}{T_i}(\rho_i + \epsilon))(y_i(t)) + (\rho_i + \epsilon)(\frac{\Delta t}{T_i}\sigma(x_i) + \frac{\Delta t}{T_i}I_i) \tag{2.21}$$

We also note that

$$\frac{dy_i}{d\epsilon} = -\Delta t y_i + \Delta t \sigma(x_i) + \Delta t I_i = \frac{dy_i}{dt}\Delta t T_i \quad . \tag{2.22}$$

Then

$$\Delta K_i = z_i(t+\Delta t)\frac{dy_i}{dt}\Delta t T_i \quad , \tag{2.23}$$

and

$$\frac{dK_i}{dt} = z_i \dot{y_i}(t) T_i \tag{2.24}$$

where $\dot{y_i}(t) = \frac{dy}{dt}$. To translate the influence of ρ to the influence of

T we use the chain rule

$$\frac{dE}{dT_i} = -\frac{dE}{d\rho_i}\frac{1}{T_i^2} = \frac{-z_i\dot{y_i}}{T_i} \quad . \tag{2.25}$$

We finally get:

$$\frac{dE}{dT_i} = -\int_t \frac{z_i \frac{dy(t)}{dt}}{T_i} dt \quad , \tag{2.26}$$

and the influence of time delays τ_{ji} on E is computed by defining a new function H where:

$$\frac{dE}{d\tau_{ji}} = \int_{t_0}^{t_f} \frac{dH_{ji}}{dt} dt \tag{2.27}$$

H_{ji} is therefore defined as:

$$H_{ji} = \frac{dE}{d\epsilon} = \underbrace{\frac{\partial E}{\partial y(t+\Delta t)}}_{=0} + \underbrace{\frac{\partial y(t+\tau_{ji})}{\partial \epsilon}\frac{\partial E(t+\tau_{ji})}{\partial y(t+\tau_{ji})}}_{II} + \underbrace{\frac{dE}{d\epsilon}}_{=0} . \tag{2.28}$$

Again, in order to get term II, the discrete function y is defined

with an infinitesimal change ϵ in the delays τ:

$$y_i(t+\Delta t,\epsilon) = \left(1-\frac{\Delta t}{T_i}\right)(y_i(t))+\frac{\Delta t}{T_i}\sigma\left(\sum_j w_{ji}y_j(t-\tau_{ji}-\epsilon)\right)+\frac{\Delta t}{T_i}I_i(t) \tag{2.29}$$

Thus, expression II becomes:

$$\frac{\partial y(t+\tau_{ji})}{\partial \epsilon} = \frac{\Delta t}{T_i}\dot{\sigma}\left(\sum_j w_{ji}y_j(t)\right)w_{ji}\dot{y}_j(t) \tag{2.30}$$

and we get an expression for H_{ji} as follows:

$$\Delta H_{ji} = -z_i(t+\tau_{ji})\dot{\sigma}(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)\frac{\Delta t}{T_i} \tag{2.31}$$

$$\frac{\partial H_{ji}}{\partial t} = -z_i(t+\tau_{ji})\dot{\sigma}(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)\frac{1}{T_i} \tag{2.32}$$

and get the time delay modification equation where the parameters correction equations are proportional to the derivative expressions:

$$\frac{\partial E}{\partial \tau_{ji}} = -\frac{1}{T_i}\int_{t_0+\tau}^{t_1+\tau} z_i(t+\tau_{ji})\dot{\sigma}(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)dt \tag{2.33}$$

All theses methods can be used for training TDRNN, however, they are extremely sensitive to the value of the training parameters

and the convergence rate is therefore very low and extremely slow. A lot of fine tuning is required for getting practical results.

2.2 Teacher forcing

Teacher forcing is a variant of gradient descent algorithms. This variant was shown to be efficient in training RNN tasks such as stable oscillations [7]. The idea is to replace the actual output of the network by the desired one, i.e. replace $y_i(t)$ by a teacher signal $f_i(t)$. This bases the future activity of the network on the teacher forced value rather then the real free running state. The dynamics of the network is thus altered during training and the corresponding learning algorithm is somewhat changed: all the derivations of teacher forced nodes with respect to the parameters will be zero. The learning algorithm is executed in the following manner:

- *Free running:* compute the free running trajectory with the current parameters.
- *Intermediate calculations:* compute the intermediate functions and other properties needed for parameter adaptation.

- *Parameters update:* compute the parameters update.

- *Teacher forcing:* Set to zero the parameter derivatives for which teacher signal exists. Replace the output value with the teacher forced value and iterate.

Although teacher forcing is a major tool for improving gradient descent algorithms, we did not include teacher forcing in our research due to the complexity it brings into indirect GD adaptation procedure.

2.3 Learning parameters

The learning parameters are the constants C_l that multiply the parameter correction equations (2.5). These constants control the rate at which the parameters are corrected. The magnitude of these multipliers is significant to the convergence of the algorithm. Choosing too small multipliers can result in very slow convergence or convergence to a local minima. In the second case, choosing large multipliers can cause the algorithm to stuck or to overflow due to an effect of "numerical resonance", where the too large correction brings the

parameters to a worse state and than the next correction is even bigger. There are several adaptive procedures to tune these parameters, one of them is to scale the parameters according to the energy sensitivity, while other uses statistical properties of the data.

2.4 Simulated annealing based training algorithms

Simulated annealing techniques are implemented for finding global minimum (or maximum) of a target function in parameter space. The method is adopted from the physical annealing procedure where a liquid is cooled down in order to obtain a minimum energy formation [11] .

These techniques where developed due to the fact that stochastic and non linear systems are extremely difficult to minimize. Stochastic methods such as simulated annealing have been found to be extremely useful tools for a wide variety of minimization problems of large non-linear systems.

Looking for minimizing a cost function $E(\eta)$ where E is a function of a vector of parameters η derived from a finite set S of possible parameter vectors. For each parameter vector indexed i, we define the set of next possible states $N(i)$ which does not include the state i, performing the following stages:

- *Selecting a candidate for the next step:* the state j is selected

out of the set $N(i)$ with a probability $g(i,j)$.

- *Computing the cost function value:* The value of the cost function for the new state $E(j)$ is computed, obtaining the cost function difference $\Delta E = E(j) - E(i)$.

- *New state decision:* If $\Delta E < 0$ the new state is adopted as the state of the system and a new iteration begins. Otherwise, the probability $h(k)$ is computed using the current temperature coefficient T_k, where k indicate the current time step of the algorithm. The new state is adopted or rejected with a probability $h(k)$ defined usually as:

$$h(k) = e^{-\frac{\Delta E}{T_k}} \tag{2.34}$$

  and the process is repeated.

The procedure is repeated while the temperature constant T_k is being slowly lowered (a quasi stationary process), and if the annealing procedure is carried out adequately [12], the system is expected to converge to the global minimum energy state. Several simulated annealing techniques were suggested over the years, among them

Boltzmann Annealing (BA) [11], Cauchy Fast Annealing (CFA) [13] and recently Adaptive Simulated Annealing (ASA) [14, 15, 16] called also Very Fast Simulated Annealing (VFSR). The basic principal of all methods is the same. The main differences are the selection of probability functions $g(\eta)$ and $h(\eta)$, resulting in significantly different convergence rates. All techniques consist of two probability functions and a cooling schedule:

- $g(\eta)$: probability density for selecting a parameter vector from the D dimensional state-space.
- $h(\eta)$: probability of accepting a new parameter vector.
- $T(k)$: Annealing "temperature" in step k, also called cooling schedule, affecting the probability for transition to a new state $h(\eta)$ and the generation probability $g(\eta)$. The annealing schedule function is actually a result of the probability functions $g(\eta)$ and $h(\eta)$.

2.4.1 Boltzmann Annealing (BA)

The BA was introduced first as a stochastic methods for implementing large dimension path integrals for statistical physics [11]. The

method was developed for coping with general minimization of cost functions. The core of the algorithm is based on the probability density driven from Gaussian Markovian systems. The Boltzmann distribution as shown in figure 2.1 is given in the following Equations:

$$g(\eta) = (2\pi T)^{\frac{-D}{2}} e^{\frac{-\Delta\eta^2}{2T}} \tag{2.35}$$

where $\Delta\eta = \eta - \eta_0$ is the deviation of η from the previous point η_0 and T is the measure of fluctuations (equivalent to temperature) of the Boltzmann distribution g in the D dimensional space η. The acceptance probability is accordingly:

$$h(\eta) = \frac{1}{1 + e^{\frac{\Delta E}{T}}} \tag{2.36}$$

where ΔE represents the cost difference between present energy state E and of the next state E^+, i.e. $\Delta E = E^+ - E$. Given $g(\eta)$ one can prove [12] that a global minimum of the problem is achieved if T is selected to decline not faster then

$$T_k = \frac{T_0}{\ln(k)} \quad . \tag{2.37}$$

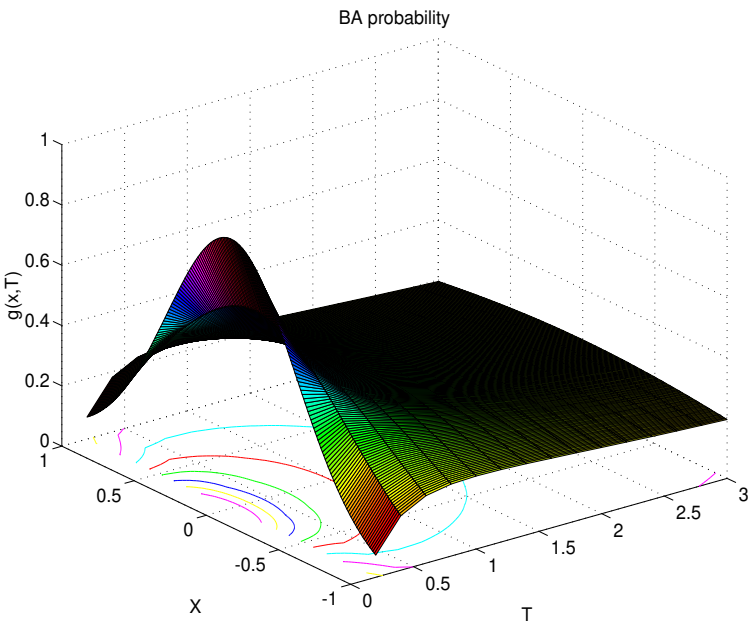


Figure 2.1: A two dimensional representation of BA probability g as a function of normalized parameter deviation and cooling temperature.

There is a proof that a global minimum of E can be achieved when implementing a proper simulated annealing procedure [12]. We will not introduce here the full proof. We will show that the probability of not generating a state η "Infinitely Often in annealing Time" (IOT) for all annealing times successive to k_0, yield zero. This proof shows that if there is a parameter-space (state-space) that contains a global minima, a proper simulated annealing algorithm will pass through this point IOT as expressed in the equations (2.38), (2.39)

and (2.40):

$$\prod_{k_0}^{\infty}(1-g_k)=0, \tag{2.38}$$

which is equivalent to:

$$\sum_{k_0}^{\infty} g_k = \infty. \tag{2.39}$$

If we select $T(k)$ as in equation (2.37) then the algorithm will pass through the minima point IOT as desired:

$$\sum_{k_0}^{\infty} g_k \geq \sum_{k_0}^{\infty} 1/k = \infty. \tag{2.40}$$

2.4.2 Cauchy Fast Annealing (CFA)

The methodology of BA can be extended to use with any function g that satisfies equation (2.38) in conjunction with a proper T decay function. Such extension might be worthwhile when the functions properties enable faster convergence rate. Using the Cauchy distribution is a step in this direction. The Cauchy distribution as shown in Figure 2.2 is defined by the following equation:

$$g(\eta) = \frac{T}{(\Delta\eta^2 + t^2)^{\frac{D+1}{2}}}. \tag{2.41}$$

This distribution has a "wide tail" in comparison with the Boltz-

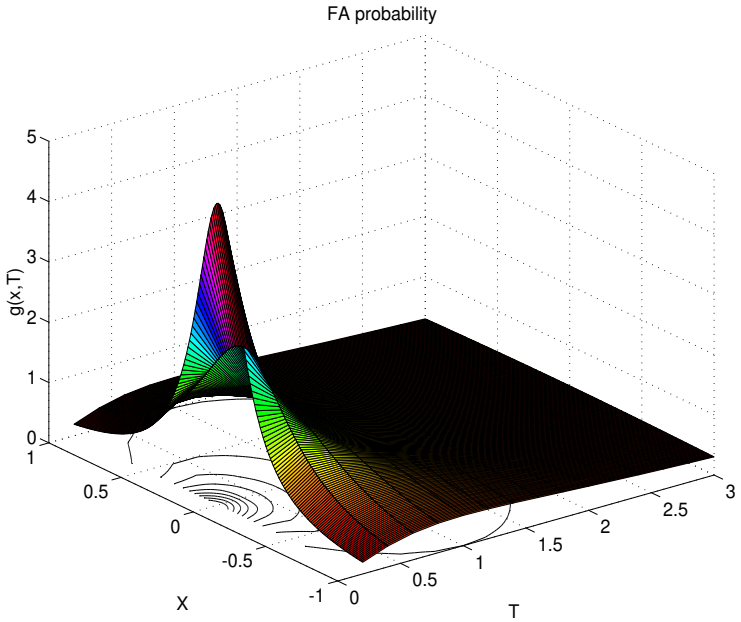


Figure 2.2: A two dimensional representation of FA probability g as a function of normalized parameter deviation and cooling temperature.

mann distribution as shown in Figure 2.5. The simulated annealing procedure using the Cauchy distribution converges to a global minimum with temperature declining not faster than

$$T(k) = \frac{T_0}{k} \tag{2.42}$$

thus,

$$\sum_{k_0}^{\infty} g_k \approx \frac{T_0}{\Delta x^{D+1}} \sum_{k_0}^{\infty} \frac{1}{k} = \infty. \tag{2.43}$$

2.4.3 Adaptive Simulated Annealing (ASA)

Adaptive simulated annealing was developed to meet three goals:

- Enable faster convergence rate.
- Obtain solution from a bounded parameter space (α) instead of unbounded space (η).
- Adapt convergence rate to parameter sensitivity.

Each of the parameters α^i at annealing time k is bounded within the range

$$\alpha_k^i \in [A_i, B_i]. \tag{2.44}$$

The parameters are generated at each new step by a random variable $y^i \in [-1, 1]$, as follows:

$$\alpha_{k+1}^i = \alpha_k^i + y^i(B_i - A_i). \tag{2.45}$$

The generating function as shown in Figure 2.3 is defined as:

$$g_T(y) = \prod_{i=1}^{D} \frac{1}{s(|y^i| + T_i) ln(1 + \frac{1}{T_i})} \equiv \prod_{i=1}^{D} g_T^i(y^i) \quad . \tag{2.46}$$

The cumulative probability distribution is

$$G_T(y) = \int_{-1}^{y^1} ... \int_{-1}^{y^D} dy^1 ... dy^D g_T(y) \equiv \prod_{i=1}^{D} G_T^{(i)}(y^i) \quad , \tag{2.47}$$

where

$$G_T^{(i)}(y^i) = \frac{1}{2} + \frac{sgn(y^i)}{2} \frac{ln(1 + \frac{|y_i|}{T})}{ln(1 + \frac{1}{T_i})} \quad . \tag{2.48}$$

y^i is generated from the uniform distribution $U[0,1]$ through a

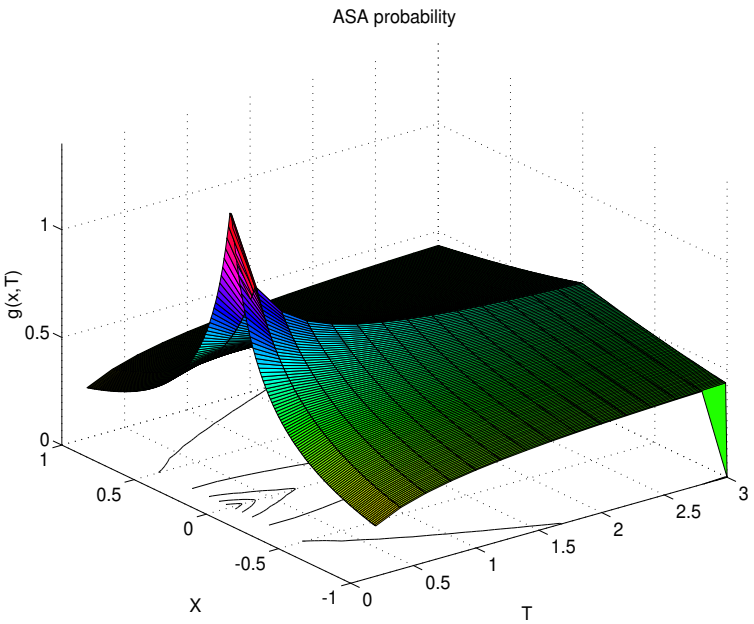


Figure 2.3: A two dimensional representation of ASA probability g as a function of normalized parameter deviation and cooling temperature.

parameter $u^i \in U[0,1]$:

$$y^i = sgn(u - \frac{1}{2})T_i \left[(1 + \frac{1}{T_i})^{|(2u^i - 1)|} - 1 \right] \tag{2.49}$$

For the annealing schedule

$$T_i(k) = T_{0i} e^{(-c_i k^{1/D})} \tag{2.50}$$

a global minima can be obtained by:

$$\sum_{k_0}^{\infty} g_k \approx \sum_{k=0}^{\infty} \left[\prod_{i=1}^{D} \frac{1}{2|y^i|c_i} \right] \frac{1}{k} = \infty. \tag{2.51}$$

Control over c_i is done through the relations

$$m_i = c k_{f_i}^{\frac{1}{D}} \tag{2.52}$$

$$k_{f_i} = e^{n_i}$$

$$T_{fi} = T_{0i} e^{-m_i} \tag{2.53}$$

$$c_i = m_i e^{\frac{-n_i}{D}}$$

The acceptance function h is of the form

$$h(\alpha) = e^{\frac{-\Delta E}{T}} \tag{2.54}$$

When implementing multidimensional search one faces different sensitivities of the cost function with respect to the parameters α_i. It is more useful to search areas sensitive to a parameter with a greater resolution. To accomplish this goal, the algorithm must rescale the annealing time k in respect to the parameters. This action is called "Re-Annealing" or "Adaptation" and is the reason why the algorithm is called adaptive simulate annealing. The adaptation occurs during the annealing procedure in terms of the energy sensitivities of the cost function at the last minimum resolved:

$$s_i = (A_i - B_i)\frac{\partial E}{\partial \alpha^i} \tag{2.55}$$

Then the annealing time k is rescaled using the maximum sensitivity $s_{max} = max(s_1 ... s_i ... s_D)$ as follows:

$$k_i' = \left(\frac{ln(\frac{T_{i0}}{T_{ik}} \frac{s_{max}}{s_i})}{c_i} \right)^D . \tag{2.56}$$

At the beginning of the process T_{i0} is set to unity. The criteria for the algorithm to stop is a low temperature value T_∞ a value that is similarly rescaled.

Parameter	BA	CFA	ASA
Probability $g(\eta)$	$(2\pi T)^{-\frac{D}{2}} e^{-\frac{\Delta\eta^2}{2T}}$	$\frac{T}{(\Delta\eta^2+T^2)^{\frac{D+1}{2}}}$	$\left(\prod_{i=1}^{D} \frac{1}{(2\lvert y_i\rvert+T_i)(1+1/T_i)}\right)$
Energy choose $h(\eta)$	$\frac{1}{1+e^{\frac{\Delta E}{T}}}$	$\frac{1}{1+e^{\frac{\Delta E}{T}}}$	$e^{\frac{-\Delta E}{T}}$
Convergence	$T(k) = T_0/ln(k)$	$\frac{T_0}{k}$	$T_0 e^{-c_i k^{1/D}}$
Re-Annealing	NONE	NONE	FEATURE

Table 2.1: A comparison between different simulated annealing techniques parameters: The probability g to choose next state, the energy threshold probability h and the cooling schedule that ensures convergence $T(k)$.

2.4.4 Summary

Table 2.1 compares the probability functions and convergence rates used for the main simulated annealing techniques. Here $\Delta\eta \equiv \eta - \eta_0$, ΔE represents the energy difference between two states, y_i is a random number (between -1 and 1) generating the new parameter set α and c_i is a parameter that controls the annealing schedule [14]. Figure 2.5 shows graphically the difference between the probability distributions $g(x)$ of the different SA methods while Figure 2.4 com-

pares the cooling schedule of the three SA methods. Though all of these methods had been considered as possible training techniques, the only one that was actually tested was ASA since it offers a useful feature for tackling the sensitivity of the training process to time delays modifications. This feature, unique for the ASA algorithm, is the reannealing procedure. This procedure changes the resolution of search (by an adaptation of an individual cooling schedule for each parameter) in proportion to the sensitivity of the energy function in respect to each parameter. The Adaptive Simulated Annealing (ASA) algorithm was found to be a robust and stable algorithm to train the time delayed neural network to perform various tasks showing both computation efficiency (most applications of network training were implemented on a 486 PC platform) and stability.

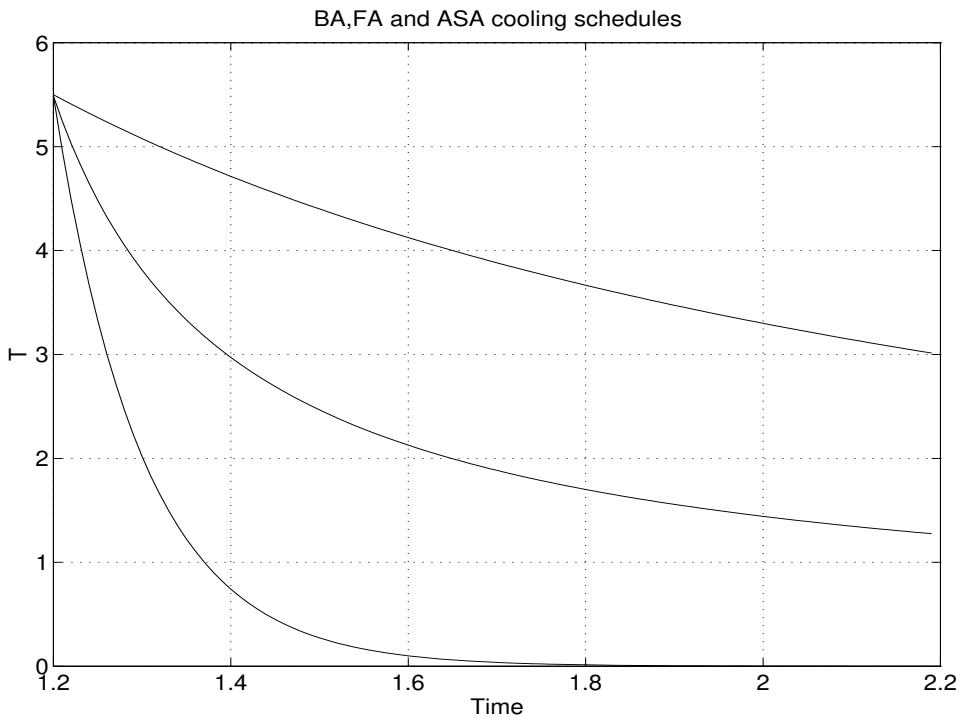


Figure 2.4: A comparison between cooling schedules of three simulated annealing algorithms: The regular Boltzmann Annealing (BA) (top), the Cauchy Fast Annealing (CFA or FA) (middle) and the Adaptive Simulated Annealing (ASA) (bottom). One can notice how a faster schedule is achieved while selecting a suitable generating probability g.

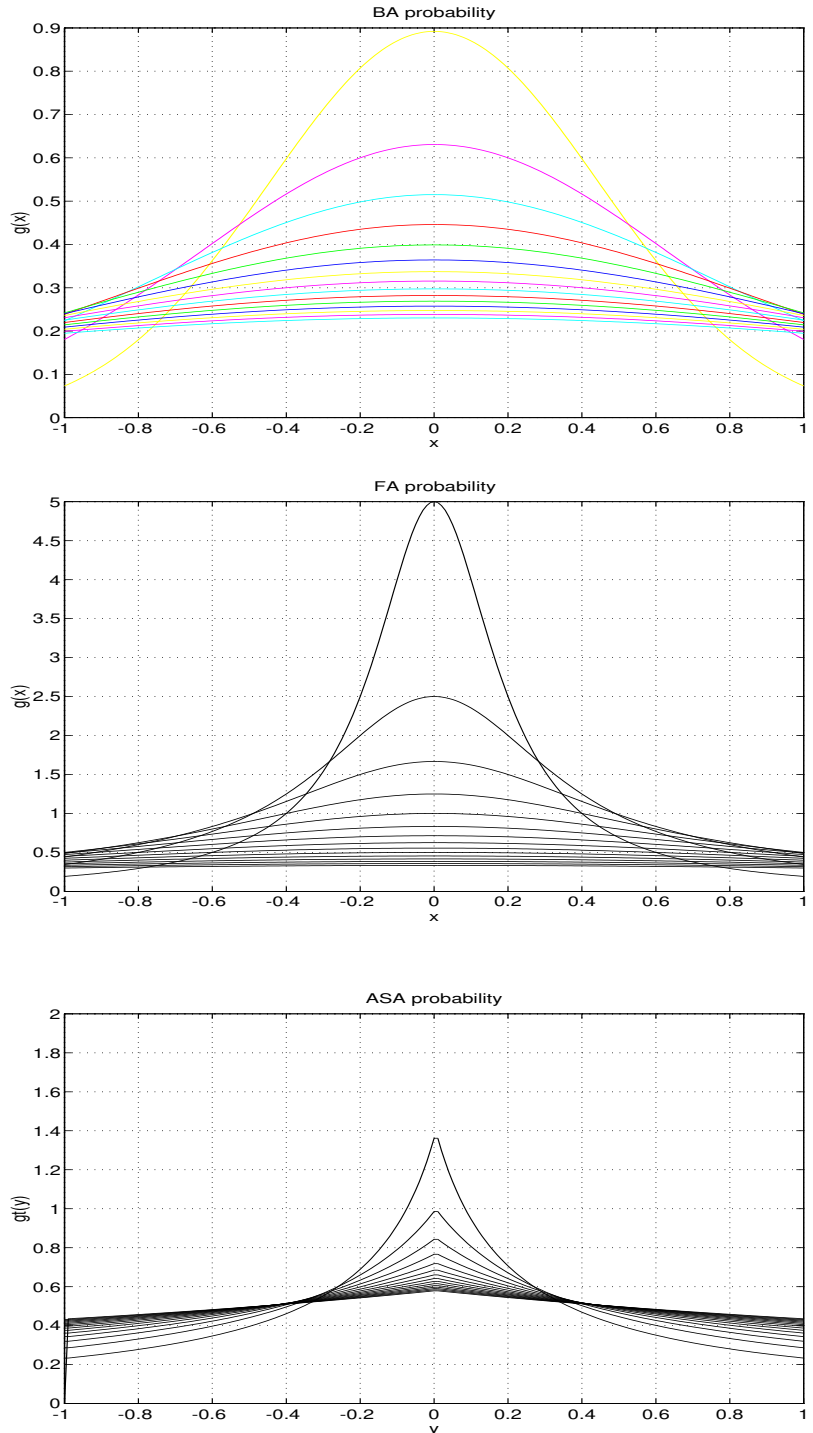


Figure 2.5: A comparison between probability functions $g(x)$ of three simulated annealing algorithms: The regular Boltzmann Annealing (BA), the Caushy Fast Annealing (CFA or FA) and the Adaptive Simulated Annealing (ASA). Note that an annealing function with thicker "wings" enable faster training procedure as shown in figure 2.4. The graphs also show the change in the probability distribution as temperature decreases, in this case between 0.2 and 3.

Chapter 3

Implementation

3.1 Task description

The algorithms were tested on several tasks chosen to demonstrate the main advatages in using synaptic time delays. The tasks are of different complexity and an effort was made to implement them using minimum size networks. The tasks are:

- *The oscillator* - The task here was to force a neural network to oscillate at a certain frequency .
- *The circle* - This problem is a common benchmark problem[5, 8] where a second oscillation with a phase difference of $\pi/2$ with

respect to the first one is required as well. Some of the teacher signals for the various tasks are introduced in Figure 3.1.

- *The eight shape* - The eight shape problem, a common benchmark problem as well, requires some sort of internal memory (internal representation) deciding on the direction of the trajectory at the cross-point, where the trajectory has two options.

- *The quadrangle* - The quadrangle problem shows the ability of TDRNN to produce non derivable trajectories. The quadrangle trajectory requires jigsaw shaped signals which are virtually impossible to create in conventional RNN framework.

- *The Bi-directional Single Trace (BST)* - The BST problem shows the ability to produce different types of periodic signals with non-derivable end points and a continuous memory for the direction of the trajectory. Here the network is required to produce the "camel-back" shaped function in one of its output neurons, this function can not be created using a conventional RNN.

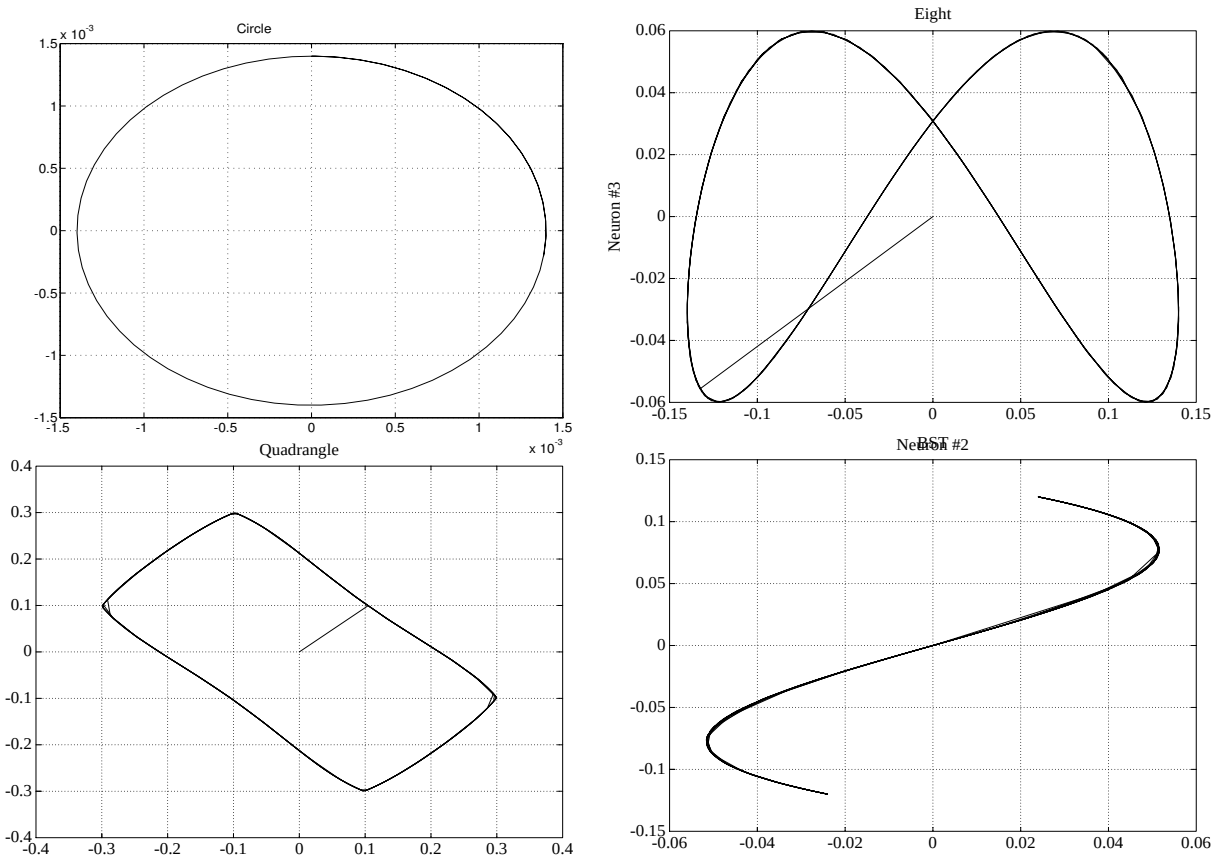


Figure 3.1: Some of the tasks teacher signals. The circle, the eight pattern, the square and the justice pattern. The patterns are shown as X-Y plots where X values are output of one neuron and Y are output of the other.

3.2 Computer resources

While extending training algorithms to include time delays, both memory (especially for non causal algorithm) and computation time (multi dimensional energy functions) increased significantly. A comparison between the memory and computation time requirements for the various methods are examined in the following subsections.

Algorithm	TDRNN Mem. alloc.	RNN
Indirect Deriv.	$\mathcal{O}(3NS + 2N^2 + N)$	$\mathcal{O}(4 + 2N^2)$
Direct Deriv.	$\mathcal{O}((2N^3 + N^2)S)$	$\mathcal{O}(2N^3 + N^2)$
ASA	$\mathcal{O}(2N^2 + N + LN)$	$\mathcal{O}(2N^2 + N + LN)$

Table 3.1: Memory consumption of different training algorithms. The memory is stated in terms of the network size N, the trajectory length L and the time delay deepness S.

3.2.1 Memory allocation

The memory requirements for implementing a training algorithm for a network of size N (neurons), a trajectory of length L (time steps) and a maximum time delay S, are shown in Table 3.1. It is obvious that the time delay causes a significant increase in memory allocation though for the configurations examined in this thesis we

have not encountered any practical problems.

3.2.2 Time consumption

The estimated computation time consumption relates to the additional computation time per iteration. Table 3.2 shows a comparison between the training algorithms in this aspect. Since the number of iterations can not be determined in any of the methods, the only comparison that can be done is the time required to accomplish one iteration, showing the complexity of the calculation.

Algorithm	TDRNN Time per Iter.	RNN
Indirect Deriv.	$\mathcal{O}(LN^3S^3)$	$\mathcal{O}(LN^3)$
Direct Deriv.	$\mathcal{O}(LN^4S)$	$\mathcal{O}(LN^4)$
ASA	$\mathcal{O}((LS+1)N^2)$	$\mathcal{O}((L+1)N^2)$

Table 3.2: Time consumption of different training algorithms per iteration. Expressed in terms of the network size N, then trajectory length L and the time delay depth S.

One can easily notice the effect of time delays on the computation time. In practice, since convergence was very fast we haven't encountered computation time consumption problems in spite of the increase in the complexity of the training.

3.3 Execution overview

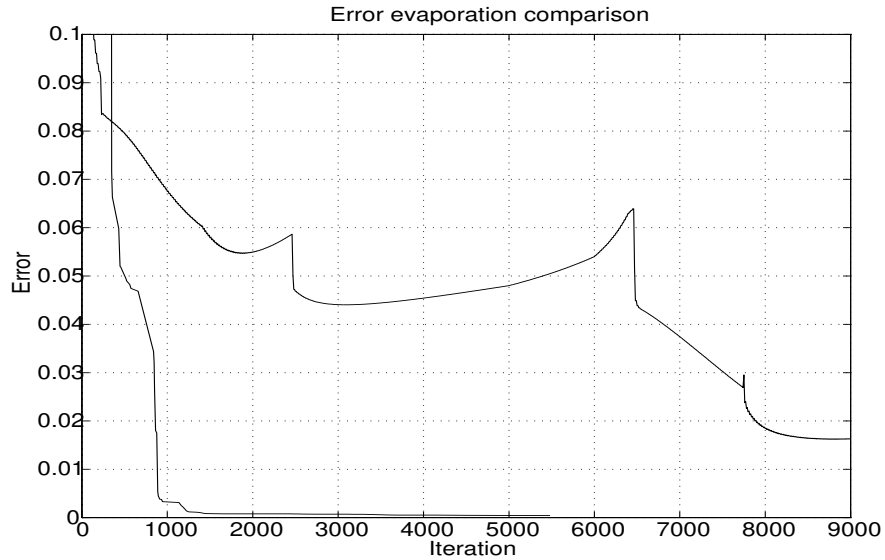


Figure 3.2: Convergence behavior of two types of algorithms. The left graph represents a typical behavior of the ASA algorithm. The right graph represents the behavior of gradient decent algorithm. One notices several irregularities in the GD graph, resulting from the rescaling procedure in the algorithm designed to tune the various parameters according to their sensitivities.

All of the tasks introduced previously where implemented using a TDRNN with less then four neurons. All three algorithms where implemented and tested. Both indirect derivation and direct derivation showed significant instability solving different problems as well as extensive tuning requirements. We therefore developed a special tuning procedure for tackling the problem. The ASA algorithm, on the other hand, seems to be more reliable, showing robustness and stability with no tuning at all. Figure 3.2 shows a comparison be-

tween typical behaviors of the algorithms in the course of training. The trained TDRNN shows a significant capability improvement over conventional RNN, enabling smaller nets to produce similar trajectories to those obtained by conventional techniques as well as trajectories that cannot be produced by conventional RNN techniques. As an example, the quadrangle and BST tasks emphasize the capability of TDRNN to produce non derivable trajectories. The tables below show a comparison between TDRNN and RNN performance for the "circle" and "eight" problems. TDRNN training was

Network type	Circle shape task	Eight shape task
TDRNN	3 neurons 6 connections	3 neurons 7 connection
RNN [5]	7 neurons 21 connections	13 neurons 78 connections
RNN [8]	7 neurons 21 connections	7 neurons 21 connections

Table 3.3: Network size comparison. One notices that TDRNN requires significantly less neurons and connections.

performed using ASA while RNN data was taken from previous works[5, 8] using the indirect approach for gradient decent method. The comparison is based on the following criteria:

- *Network size:* Comparing both number of neurons and the number of connections as presented in Table 3.3. It is easy to see that TDRNN requires significantly less neurons and connections for performing similar tasks.

- *Computation time:* The number of epochs it takes to complete the training procedure. The completion of the training procedure is determined by a minimal error threshold criterion. Table 3.4 compares the convergence time of the training algorithms. Note that the parameter measured here is epochs which means number of iterations of the algorithm's kernel.

Training Method	Circle shape task	Eight shape task
TDRNN ASA algorithm	1800 epochs	2200 epochs
TDRNN Indirect GD	60000 epochs	Not Achieved
RNN [5] Indirect derivation algorithm	12000 epochs	20000 epochs
RNN [8] Indirect derivation algorithm (modified)	800 epochs	1000 epochs

Table 3.4: Convergence time benchmark. One notices that the ASA algorithm converges time is of the same scale as other algorithms implementing regular RNN while gradient decent algorithm for training TDRNN require longer training procedure and might fail training.

The trained networks produced the trajectories presented in Figure 3.3. In order to show the capabilities both of the algorithm and of the TDRNN architecture, we examined them on a set of training tasks mentioned previously. Following is a description of the training results for the various tasks:

- *Oscillator* - Training the network to produce an oscillator requires two neurons. The key for creating an oscillation is a local feedback loop with time delay for the oscillating neuron. The smoothness of the oscillation is influenced mostly by the time constant T_i, controlling the magnitude of signal derivative. Using ASA the algorithm coincided with the required pattern after 1200 epochs.

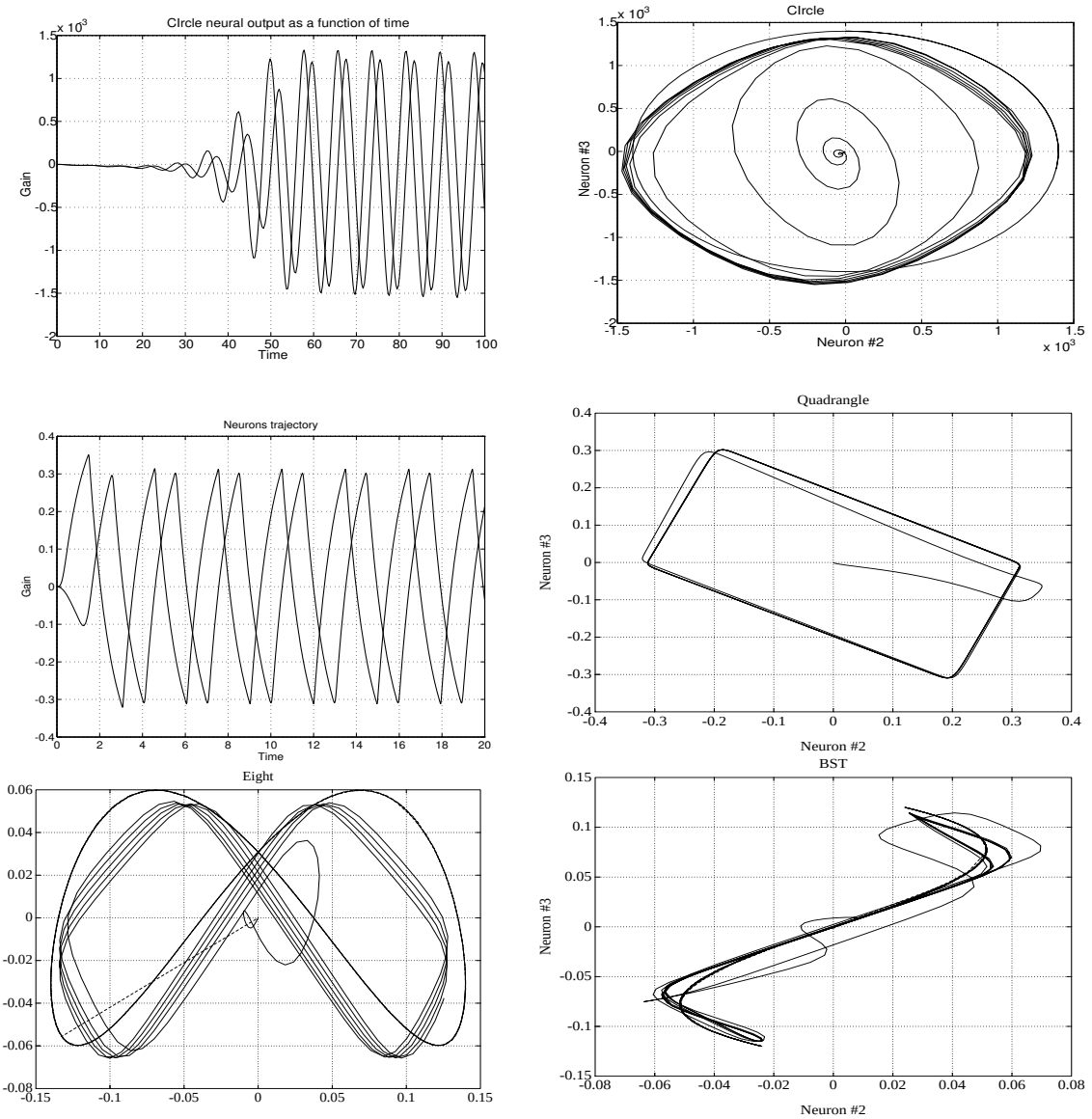


Figure 3.3: The TDRNN output signals: The top graphs show the circle pattern as a function of time (left) and as an X-Y plot (right). The middle graphs show the quadrangle pattern as a function of time (left) and as an X-Y plot (right). The bottom graphs show the eight shape (left) and the BST (right).

- *Circle* - A circle trajectory was obtained using three neurons, reflecting a significant decrease in network size in comparison to conventional RNN where seven neurons were required.

- *Eight shape* - The eight shape is generated by two related internal frequencies. Using TDRNN configuration, the eight shape was achieved using three neurons. It took two oscillation cycles before generating the proper signal.

- *Quadrangle* - The quadrangle task shows TDRNN capability to produce non derivable trajectories. The three neurons network stabilizes after two cycles. The main feature that generates this non derivable signal is clearly the time delay property.

- *BST* - This non standard repeated pattern is learned by the network showing the capability of TDRNN to store the direction of a repeated trajectory with non derivable end points using time delays. It has been previously shown[10] that time delayed networks can learn to produce such signals.

Table 3.5 summarizes the configurations and the training results for the various tasks:

The ASA algorithm reaches the vicinity of the desired parameters quite fast though the final convergence is somewhat longer. This behavior is typical for ASA; an example for the decrease in training

error (equation 1.2) is shown in Figure 3.2.

It is also important to note that eliminating optional connections in the network contributes to a faster training procedure (if no external suppression of redundant weights is implemented the training procedure carries it out by enlarging the time delay to the maximum or reducing the weight itself, requiring additional computation time).

Task	Network Size	Connections	Epochs
Oscillator	2	2	1200
Circle	3	6	1800
Eight Shape	3	7	2200
Quadrangle	3	7	1900
BST	3	7	2400

Table 3.5: ASA training performance summary. The ASA algorithm was proven as a stable algorithm for solving various tasks.

3.4 Network behavior

Several typical behaviors for both the training procedure and the trained nets themselves have been noticed :

- *Signal mismatch*

  In all the tasks trained a slight mismatch between the output signal and the required output was observed. This mismatch was observed also in conventional RNN networks[8] and training procedures and is a result of the dynamics of the network. One can assume that larger networks will result with a smaller mismatch.

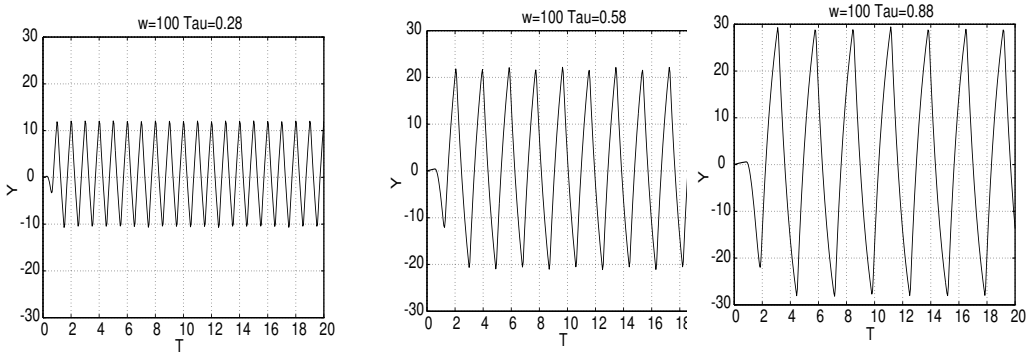


Figure 3.4: The influence of local feedback with time delay on oscillation, the graphs show different solutions of $\frac{dy}{dt} = -y - w\sigma(y(t-\tau)) + h(0)$ for $\tau = 0.28, \tau = 0.58, \tau = 0.88$ and $w = 100$. One notices the changes in local oscillation frequency and amplitude.

- *Frequencies and amplitude*

  There is a certain coupling between the amplitude and the frequency of a given neuron (especially in local feedback states). This phenomena results from the dynamics equation of the sys-

tem (equation 1.1) In an oscillation evolving due to a local feedback with time delay (the local frequency is proportional to $1/\tau_{ii}$), the time delay affects the amplitude of the local oscillation. We can show the phenomena on a single neuron with an equation of motion of the form

$$\frac{dy}{dt} = -y - w\sigma(y(t-\tau)) + h(0) \tag{3.1}$$

Here $-w$ is the local feedback and τ is the time delay. The behavior of such system with respect to the time delays is shown in Figure 3.4. One notices that while increasing τ the frequency of the local oscillation decreases and the amplitude increases.

- *Perturbation reaction*

  The sensitivity of the trained net to changes in the input signal was examined for the quadrangle task. Modifications in the input signal $I(t) = h(0)$ were implemented in several ways:

  - *DC gain*

    The input gain was modified: $I(t) = 2h(0)$,$I(t) = h(0) - h(0.01)$ The only effect on the trajectory was on the tran-

sient behavior (until the second oscillation cycle). The free neuron [1] was affected, changing it's amplitude. Figure 3.5 shows the network behavior.

- *Frequency (AC)*

  Sinusoidal signal was added to the input in two cases $I(t) = h(0) + 0.5\sin(t)$ and $I(t) = h(0) - \sin(10t)$. Also here, the only effect on the trajectory was on the transient behavior (until the second cycle). The free neuron was affected, and changed it's behavior. Figure 3.5 shows the network behavior.

The input signal in the quadrangle task seems to generate the oscillation, affecting only the transient behavior of the oscillation. The oscillation itself is mainly controlled by the other neurons. Similar behavior in conventional RNN was mentioned in previous works [5, 8]. This phenomena is consistent as long as the input signal does not cause the system to saturate. Saturation in this case occurs when the average magnitude of the

[1] The free neuron is the neuron that need not to comply with any teaching rule, In our case Neurons 2 and 3 have to produce a trajectory and Neuron 1 is free.

input signal raises over 30.0.

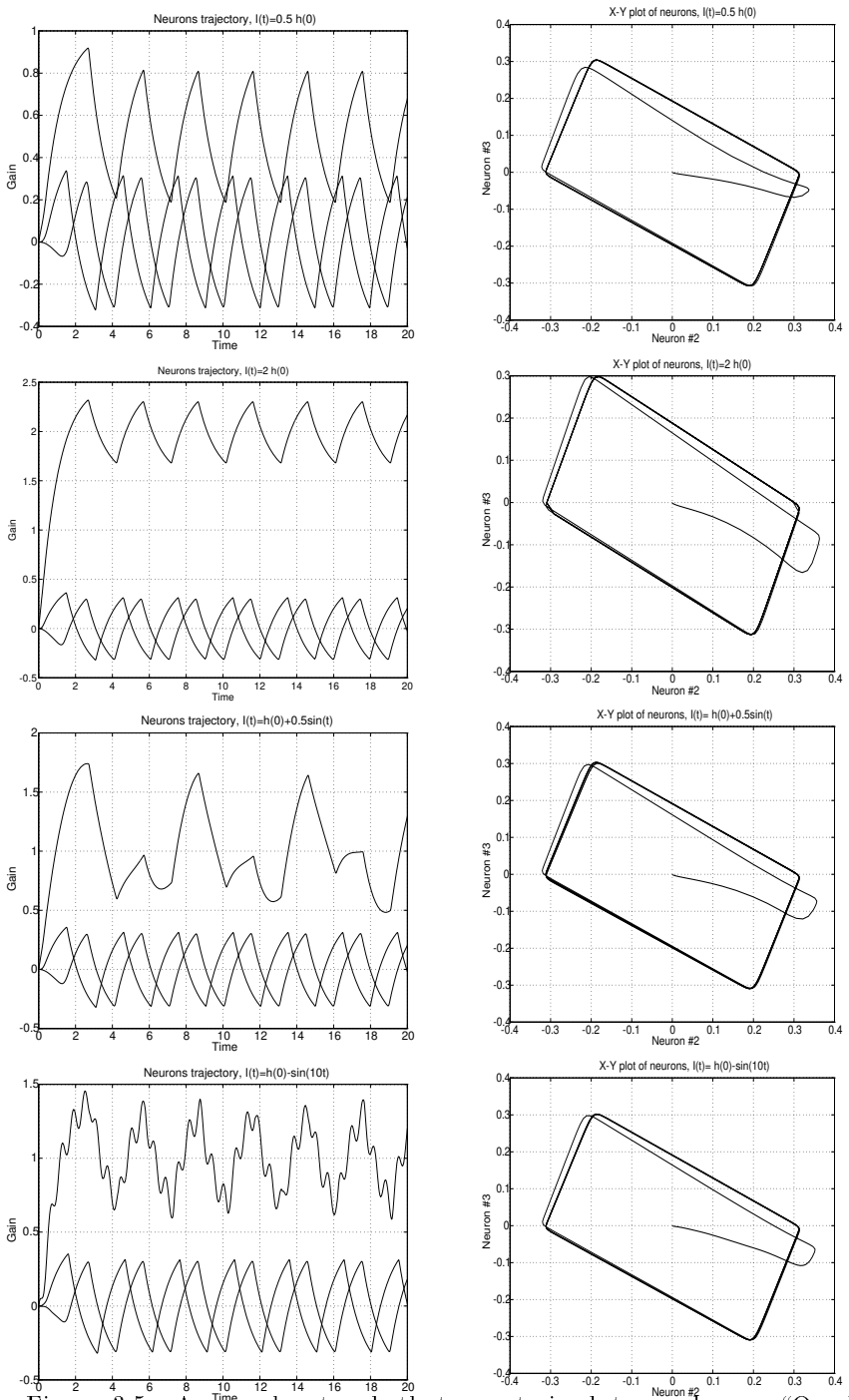


Figure 3.5: A neural network that was trained to produce a "Quadrangle" trajectory with an input $I(t) = h(0)$ response for input perturbations. The original trajectory is shown in Figure 3.3. The graphs (from top to bottom) show the response of the TDRNN for the following perturbations in the input signal: $I(t) = h(0)+0.5\sin(t)$ and $I(t) = h(0)-h(0.01)$, $I(t) = h(0)+0.5*\sin(t)$ and $I(t) = h(0) - \sin(10t)$. The right graphs show the trajectory on an X-Y plot while the left show the response as a function of time. On the left graphs one notices the changes in the behavior of the free neuron while after attaining the oscillation the other neurons maintain a stable trajectory.

Chapter 4

Other TDRNN configurations

The network configuration used up to now is the extension of the configurations that where implemented in previous works [5]. There is no proof, however, that this configuration is optimal. There might be other configurations that perform better. In this section we will offer two alternatives for the neural network structure. Note that all the alternative configurations are modifications of the general configuration introduced in the beginning of this work and can learn to perform the same tasks. The idea behind the suggestion of such

configurations is to examine the connection betwwen the network dynamics and its ability to perform certain tasks.The first configuration which we shall refer to as (MODEL 1) is a more nonlinear system where the neural function operates also on the external input to the neural system as shown in equation (4.1):

$$\dot{y}_i(t) = \frac{1}{T_i}\left[-y_i + \sigma\underbrace{\left((\sum_j w_{ij}y(t-\tau_{ij})) + I_i\right)}_{x_i}\right] \tag{4.1}$$

The second alternative is a more linear configuration which we shall refer to as (MODEL 2) where the nonlinearity is applied to the various output values as shown in equation (4.2):

$$\dot{y}_i(t) = \frac{1}{T_i}\left[-y_i + \left(\underbrace{(\sum_j w_{ij}\sigma(y(t-\tau_{ij})))}_{x_i} + I_i\right)\right] \tag{4.2}$$

Figure 4.1 shows graphically the dynamics of the different models. It is not the scope of this work to find the best network configuration. We will rather introduce appropriate training algorithms and show the ability of the alternate configurations to learn and perform tasks.

We will view the new configurations from two aspects: First, from the performance point of view, different configurations might be more suitable for certain types of tasks. Second, the adaptation procedure where different configuration might lead to a quicker and more efficient training procedure.

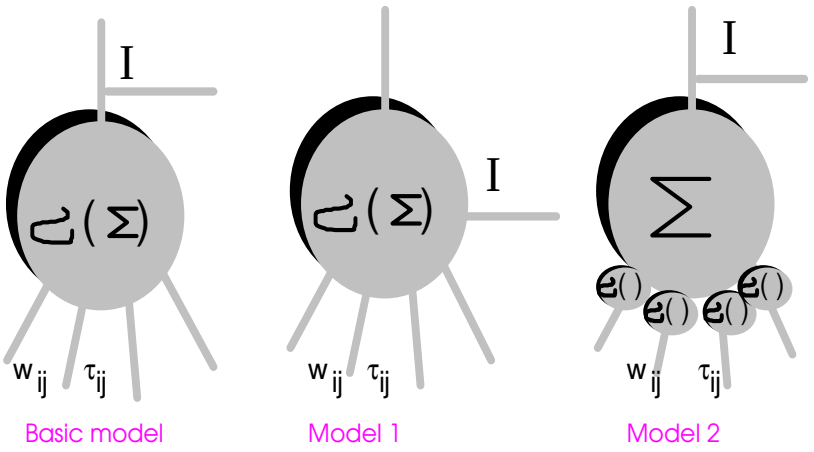


Figure 4.1: A comparison between the structure of three different models of TDRNN: The basic model (left), MODEL1 (middle) and MODEL2 (right).

We will derive the two gradient descent algorithms for both configurations. The Stochastic approach is the same as introduced in section 2 and the ASA algorithm can be used as is for the new configurations.

4.1 Direct gradient descent for MODEL 1

The training algorithm is achieved by deriving equation (4.1) with respect to the various parameters in the same manner as we did in equation (2.2) to obtain the following terms: p_{ij}^k, q_i^k and r_{ij}^k, representing the sensitivity of the energy function to the various parameters. Taking the derivatives with respect to the various parameters, one obtains the following equations:

$$\begin{aligned} T_k \frac{dp_{ij}^k}{dt} &= -p_{ij}^k + \dot{\sigma}(x_k) \sum_l w_{lk} p_{ij}^l \\ T_k \frac{dq_i^k}{dt} &= -q_i^k + \dot{\sigma}(x_k) \sum_l w_{lk} q_i^l - \dot{y}_k \\ T_k \frac{dr_{ij}^k}{dt} &= -r_{ij}^k + \dot{\sigma}(x_k) \left[w_{ij} \frac{dy_i}{dt}(t - \tau_{ij}) \delta_{jk} - \sum_l w_{lk} r_{ij}^l (t - \tau_{lk}) \right] \end{aligned} \tag{4.3}$$

The parameters correction process is carried out by propagating the above equations forwards in time as we did with equations (2.5) and computing the sensitivity of the enrgy function to the parameters, thus correcting the parameters in the process of learning.

4.2 Direct gradient descent for MODEL 2

The training algorithm is achieved by deriving equation (4.2) with respect to the various parameters in the same manner as we did in Equation (2.2). to obtain the following terms: p_{ij}^k, q_i^k and r_{ij}^k. Also here, one obtains the following equations for the various parameters:

$$\begin{aligned} T_k \frac{dp_{ij}^k}{dt} &= -p_{ij}^k + \sum_l w_{kl} \dot{\sigma}(y(t-\tau_{kl})) p_{ij}^k \\ T_k \frac{dq_i^k}{dt} &= -q_i^k - \dot{y}_k \\ T_k \frac{dr_{ij}^k}{dt} &= -r_{ij}^k - \sum_l w_{ij} \dot{\sigma}(y(t-\tau_{ij})) r_{ij}^k (t-\tau_{ij}). \end{aligned} \tag{4.4}$$

The parameters correction process is obtained by propagating the above equations forwards in time as we did in equation set (2.5) and computing the sensitivity of the energy function to the parameters.

4.3 Indirect gradient descent for MODEL 1

This training algorithm is very close to the original algorithm presented in the beginning of this work. The major change is a new

definition of $x_i(t)$ in equation (4.1). Except for this change the training algorithm is identical:

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= -\int_{t_0+\tau_{ji}}^{t_1+\tau_{ji}} \frac{z_i(t+\tau_{ji})\dot{\sigma}(x_i(t))y_j(t)}{T_i}dt, \\ \frac{dE}{dT_i} &= -\int_t \frac{z_i\frac{dy(t)}{dt}}{T_i}dt \end{aligned} \tag{4.5}$$

and the influence of time delays τ_{ji} on E is:

$$\frac{\partial E}{\partial \tau_{ji}} = -\frac{1}{T_i}\int_{t_0+\tau_{ji}}^{t_1+\tau_{ji}} z_i(t+\tau_{ji})\dot{\sigma}(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)dt \tag{4.6}$$

4.4 Indirect gradient descent for MODEL 2

For the extended linear model there is a significant change in the evolutionary process of the algorithm because of the change in the dynamic behavior. The derivation of the algorithm is done in a similar way and is introduced in appendix C. The training equations are based on the redefinition of the back propagated function as

follows:

$$z_i(t) = e_i(t)\Delta t + z_i(t+\Delta t)(1+\frac{\Delta t}{T_i}) \tag{4.7}$$
$$+\sum_j z_j(t+\tau_{ij})\frac{\Delta t}{T_j}\left(\sum_l w_{li}\dot{\sigma}\left(y_l(t+\tau_{li})\right)+I_i(t)\right)$$

and the sensitivity equations are:

$$\frac{\partial E}{\partial w_{ji}} = -\int_{t_0+\tau_{ji}}^{t_1+\tau_{ji}} \frac{z_i(t+\tau_{ji})\left(\sum_j \dot{\sigma}(y_j(t))\right)}{T_i}dt \tag{4.8}$$

$$\frac{dE}{dT_i} = -\int_t \frac{z_i\frac{dy(t)}{dt}}{T_i}dt \tag{4.9}$$

$$\frac{\partial E}{\partial \tau_{ji}} = -\frac{1}{T_i}\int_{t_0+\tau}^{t_1+\tau} z_i(t+\tau_{ji})(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)dt \tag{4.10}$$

4.5 Implementations

Due to the poor performance of GD algorithms with our basic models (1.1) , we concentrated in using the ASA algorithm for implementing the same benchmark tasks that were used in the basic models. The results are described in Table 4.1. Figure 4.2 shows the convergence of the ASA algorithm training the three TDRNN models to perform our benchmark tasks. One can notice that the ASA algorithm was able to train all three of the models to perform the tasks.

The training performance of the three models differs for different tasks:

Task	Basic Model	Model 2	Model 1
Circle	3 Neurons 6 Connections 2000 Epochs	3 Neurons 6 Connections 18000 Epochs	3 Neurons 6 Connections 4000 Epochs
Eight Shape	3 Neurons 7 Connections 2200 Epochs	3 Neurons 7 Connections 750 Epochs	3 Neurons 7 Connections 1100 Epochs
quadrangular	3 Neurons 7 Connections 1600 Epochs	3 Neurons 7 Connections 2200 Epochs	3 Neurons 7 Connections 1200 Epochs
BST	3 Neurons 7 Connections 4000 Epochs	3 Neurons 7 Connections 1000 Epochs	3 Neurons 7 Connections 3200 Epochs

Table 4.1: ASA training performance summary for different TDRNN models. Note that the performance of the basic model is slightly different from what was presented in chapter 3. This is due to different initial parameters that were chosen in order to make equal tests for the three models.

- *Circle:* both MODEL 1 and MODEL 2 reached an area in the parameter space that is close to the global minima faster than the basic model, yet the final minima search was faster in the basic model.

- *Quadrangle:* All three models behave in a similar manner. MODEL 1 reached the minima area faster than the others.

- *Eight figure:* both MODEL 1 and MODEL 2 behaved in the same manner and reached the global minima area faster than the original model.

- *BST:* MODEL 2 proved superiority over MODEL 1 and the basic model which has the slowest convergence rate.

4.6 Conclusions

from the limited configurations and training tests examined, one can conclude that:

- All of the models of TDRNN are capable of learning to perform the benchmark tests examined in this work.

- We find that different TDRNN configurations performs differently for different problems, for example, MODEL 2 performs better for the BST problem and the quadrangular problem, while the basic model is better with the circle task.

- Different TDRNN models encapsulate different qualities. These qualities influence both the training procedure and the performance envelope of the algorithm, for instance MODEL 2 has a

wide dynamic output range.

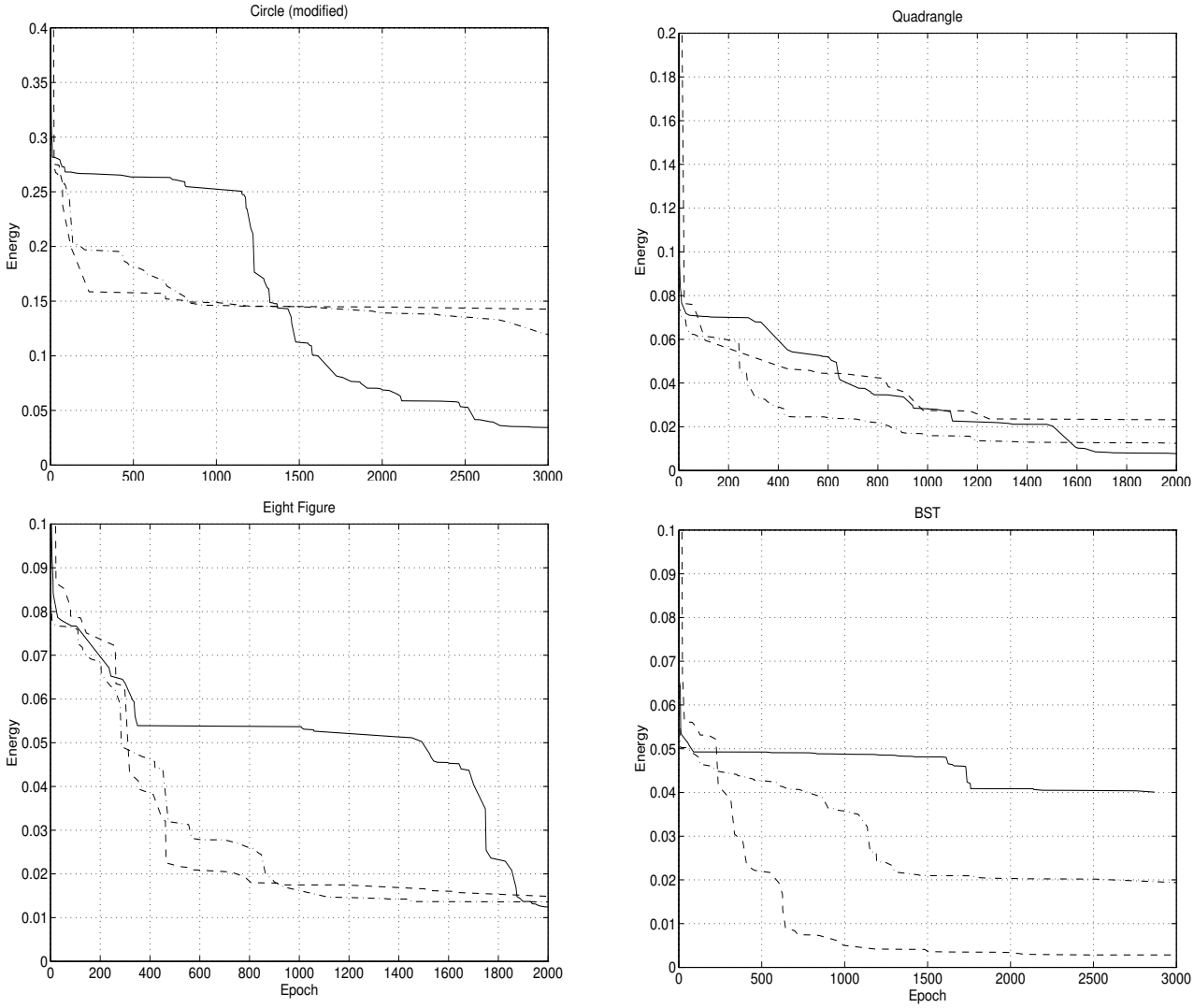


Figure 4.2: A comparison between the training schedule of three different models of TDRNN using ASA: the basic model (solid line), MODEL 1 (dot dashed) and MODEL 2 (dashed line)

Chapter 5

Summary

In this thesis we introduced training algorithms and applications of continuous neural networks with time delays. Several advantages of TDRNN with respect to conventional RNN were introduced and demonstrated. Among these advantages are:

- *Enlarging the set of trainable trajectories*: The TDRNN were able to output non derivable trajectories while RNN were able to produce only smooth trajectories.

- *Reducing network size for a given task*: The TDRNN were significantly smaller in size and in connections number than the equivalent RNN. In addition, the number of free parameters

used for performing the task was reduced in comparison with RNN.

- *Enlarging internal states space*: Due to the addition of time delays TDRNN of same size (neurons number) as RNN have larger internal state space.

- *Improving generalization capabilities*: The addition of time delays (thus, enlarging the state space) is not equivalent to an addition of neurons to RNN. The great influence of time delays on the behavior of the network causes the adaptation procedure to discard large portions of the state space that are irrelevant to the desired solution, thus, completing the adaptation procedure faster and obtaining a more general solution.

Several training algorithms for TDRNN were tested. Conventional gradient descent type algorithms were found to be extremely sensitive to both the selection of training parameters and to the implementation of the training procedure itself, as well as extremely demanding in terms of computer resources. The most successful choice seems to be a simulated annealing type algorithm (ASA),

showing fast convergence and stability, enabling to train small scale TDRNN to perform rather difficult benchmark tasks on a low-end platform. Using ASA for training TDRNN leads to successful training of rather difficult tasks with significant reduction both in training time and network size than previously reported[5, 8].

Other TDRNN models were introduced and training algorithms for them were developed. The work showed the capability of different TDRNN architectures to perform various tasks as well as the apparent connection between the TDRNN topology and the performance envelope (the tasks it can be trained to perform).

Although the training procedure of TDRNN is generally slower than RNN, it results with smaller networks with better generalization capabilities. TDRNN are expected to be useful for a variety of possible applications such as: signal generators, switches, prediction networks, control systems etc. The promising results reported in this work indicate the advantages, possible use and training techniques for using TDRNN. These subjects should be explored further on in future work.

Bibliography

[1] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning representations by back-propagating errors", *Nature*, **323**,533-536, (1986).

[2] R. Rohwer, "The "Moving Targets" training algorithm", *Neural Information Processing Systems*, **2**, 558-565, (1990).

[3] A. Krogh, G.I. Thorbergsson and J.A. Hertz, "A cost function for internal representations", *Neural Infcrmation Processing Systems*, **2**, 733-740, (1990).

[4] F.J. Pineda, "Generalization of back-propagation to recurrent RNN", *Phys. Rev. Lett.*, **59**, 2229-2232, (1987).

[5] B. A. Pearlmutter, "Learning state space trajectories in Recurrent Neural Networks", *Neural Computation*, **1**, 263-269,

(1989).

[6] R.J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks", *Neural Computation*, **1**, 210-280, (1989).

[7] R. J. Williams and D. Zipser, "Experimental analysis of the RTRL algorithm", *Connections*, **1**, 87-111, (1989).

[8] N. Toomarian and J. Barhen , "Adjoint functions and temporal learning algorithms in neural networks", *Neural Information Processing Systems (Denver 1990)*, **3**, 113-120, (1991).

[9] Mittra, F. Romeo , A. Sangiovanni-Vincentelli, "Convergence and finite time behavior of simulated annealing", *Adv. Appl. Prob.*, **18**, 100-111, (1986).

[10] S. P. Day and M. R. Davenport, "Continuous time Temporal Back Propagation with Adaptable Time Delayes", *IEEE Transactions on neural networks*, **4**, 400-408 , (1992).

[11] N. Metropolis , A.W. Rosenbluth , M.N. Rosenbluth , A.H. Teller and E. Teller, "Equation of state calculations by fast

computing machines", *Journal of Chem. Phys.*, **21** , 1087-1092, (1953).

[12] S. Geman , D. Geman, "Stochastic relaxation , Gibbs distribution and the Bayasian resortation in images", *IEEE Trans. Patt. Anal. Mac. Int.* , **6**, 721-724 , (1984).

[13] H. Szu and R. Hartley, "Fast simulated annealing", *Phys. Lett A* , **122** , 157-162, (1987).

[14] L. Ingber, "Very fast simulated re-anealing", *Mathematical and Computing Modelling* , **Vol 12** , 967-973, (1989).

[15] L. Ingber and B. Rosen , "Genetic algorithms and very fast simulated anealing: a comparison", *Mathematical and Computing Modelling* , **Vol 16** , 87-100, (1992).

[16] L. Ingber, "Simulated anealing: practice versus theory", *Statistics and Computing* , **JAN** , (1993).

[17] D. Whitley, T. Starkweather and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity", *Parallel Comp.*, **14**, 347, (1990).

[18] J. Hertz, A. Krogh, R. G. Palmer , "Introduction to the theory of neural computation", **Addison-Wesley** , (1991).

[19] M. Abramowitz, I.A. Stegun, "Handbook of mathematical functions", **Applied mathematics**, **55**, 255 , (1971).

[20] C.W. Gear, "Numerical Initial Value Problems in Ordinary Diffrential Equations", **Prentice Hall**, Chapter 2 , (1971).

[21] J.R. Cash and A.H. Karp, *ACM Transactions on Mathematical Software*, **16**, 201-222, (1990).

[22] M. Feran, "Constructive and destructive networks", *Neural Computation*, **2**,198-209, (1990).

Appendix A

Numerical integration methods

Computer simulations of dynamic neural networks involved extensive use of numerical integrations. This Appendix will introduce the main numerical integration algorithm that was used throughout this work. The technique used is a predictor integration step corrector, a modified Runge Kutta algorithm of degraded 3rd degree (or shortly the RK23 algorithm). The basic RK algorithm was introduced in [19] and [20] and the adaptive step version was introduced in [21].

The algorithm solves a set of differential equations of the form

$$\frac{dy_i(t)}{dt} = F(y_1..y_i..y_N, t). \tag{A.1}$$

The algorithm computes the derivatives at each time interval and its close neighborhood and estimates the proper step size (predicts the error and corrects the time step). The computation of the next value is accomplished with the Runge Kutta 3rd order method and the prediction is implemented with Runge Kutta 2nd order method. The explicit algorithm follows:

1. Set the initial values $y_i(t = 0) = y_0$ and initial step $h = h_0$ and time $t = 0$.

2. Compute derivations dy_i/dt at time $t, t + h/2$ and $t + h$: s_{1i},s_{3i} and s_{2i} respectively.

$$s_{1i} = \frac{dy_i(t)}{dt} \tag{A.2}$$

$$s_{2i} = \frac{dy_i(t + \frac{h}{2})}{dt} \tag{A.3}$$

$$s_{3i} = \frac{dy_i(t + h)}{dt}$$

3. Compute the predictor value :

$$p_i = \frac{h(s_{1i} - 2s_{3i} + s_{2i})}{3} \tag{A.4}$$

$$\delta = |p| \tag{A.5}$$

Compute the maximum value:

$$Y_{max}(t) = MAX\,(y_1(t), y_2(t)..., y_i(t), ..., y_N(t)) \tag{A.6}$$

Compute the parameter η while using ϵ the desired integration error:

$$\eta = MAX(Y_{max}, 1.0)\epsilon \tag{A.7}$$

4. If $\delta < \eta$ then the next value is computed through

$$y_i = y_i + h\frac{(s_{1i} + 4s_{3i} + s_{2i})}{6} \tag{A.8}$$

the time is propagated forwards, $t = t + h$ and the iteration continues from stage 2.

5. If $\delta > \eta$ then a new step h is computed through

$$h_{old} = h \tag{A.9}$$

$$h_{new} = 0.9 h_{old} \frac{\tau}{\delta}^{\frac{1}{3}}$$

$$h = h_{new}$$

$$\tag{A.10}$$

The algorithm stops when the time encounters final value.

Appendix B

RNN implementations

This section briefly describes related works for training RNN. All of these works utilized various versions of gradient descent algorithms. We made an effort to try to reconstruct some of these methods for RNN and to try and adapt the methods developed in these works to deal with TDRNN. The main outcome of this effort was a better understanding of TDRNN behavior and training problems, leading us to find a better way to cope with TDRNN training tasks.

B.1 Real Time Recurrent Learning (RTRL)

The RTRL scheme was the result of direct derivation method developed [7] for training RNN. The semantics in this work is slightly different: The neural function is $\sigma(s) = (1-e^{-s})^{-1}$ and the dynamic equation is:

$$y_k(t+\Delta t) = \sigma\left(\sum_l w_{kl} z_l(t)\right), \tag{B.1}$$

where $z_k(t) = \cup(x_{k\subset input}(t), y_{k\neq input}(t))$. The neurons in this model are divided into three groups I: input, T: target, U: the rest. $\mathbf{z}$ is the neural state, $\mathbf{s}$ is the neural input, $\mathbf{y}$ is neuron output and w are the weights. This is a discrete system (unlike our basic model) and the parameter correction is computed by direct derivation:

$$p_{ij}^k = \frac{\partial y_k}{\partial w_{ij}} \tag{B.2}$$

Thus, the discrete parameter derivation is:

$$p_{ij}^k(t+\Delta t) = \dot{\sigma}(s_k(t))\left[\sum_{l\subset U} w_{ki} p_{ij}^l(t) + \delta_{ik} z_j(t)\right] \tag{B.3}$$

where $\dot{\sigma}(x) = \frac{d\sigma}{dx}$ and the parameter correction is

$$\Delta w_{ij} = \int_t \alpha \sum_{k \subset U} e_k(t) p_{ij}^k(t) dt, \tag{B.4}$$

where e is the error between the desired and current output. The on-line variation of this algorithm is called RTRL and is implemented by computing the parameter correction while the system is propagating forward in time. Williams and Zipser [7] implemented this algorithm with and without teacher forcing. Table B.1 specifies some of their results:

Task	results
Sine Wave	6 neurons 30,000 Iterations Teacher forcing
XOR	7 neurons 1,000 Iterations Teacher forcing

Table B.1: The performance of RTRL algorithm with two typical tasks as presented by Williams and Zipser.

The adaptation of this method for TDRNN was shown to be inefficient. The output of the RNN trained network is in figures B.1, B.2, B.3 and B.4.

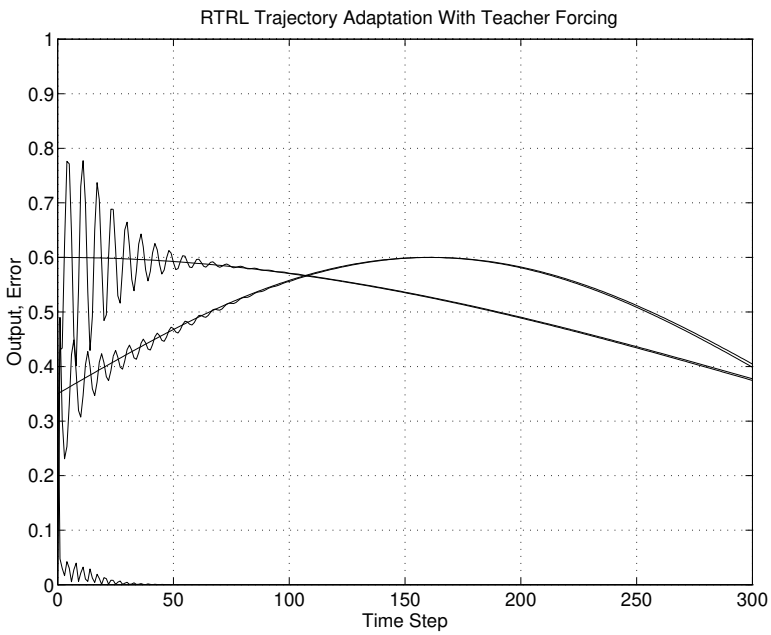


Figure B.1: RTRL training of a stable oscillation. One notices the coinciding of the trained oscillating signal over the desired signal.

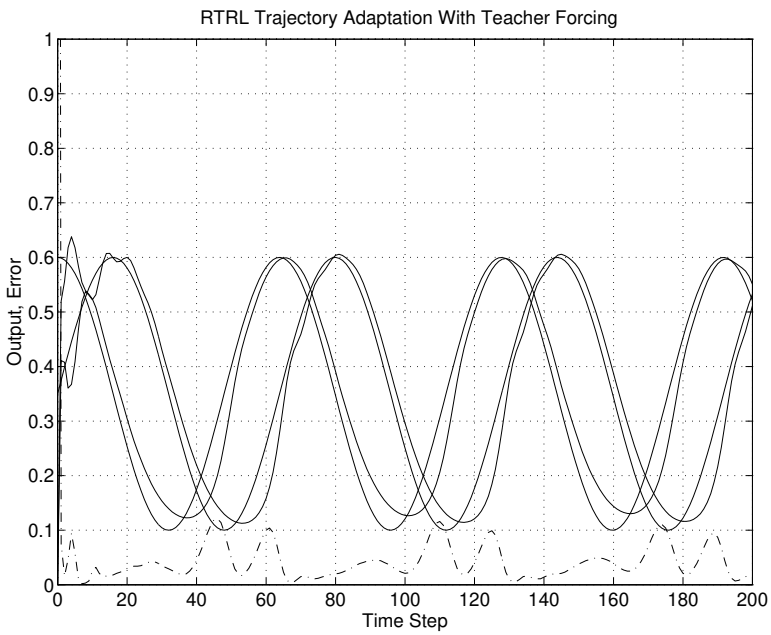


Figure B.2: RTRL training of two stable oscillations. The signals are drawn over the desired signals. The double dashed line at the bottom is the difference between the actual and desired signals.

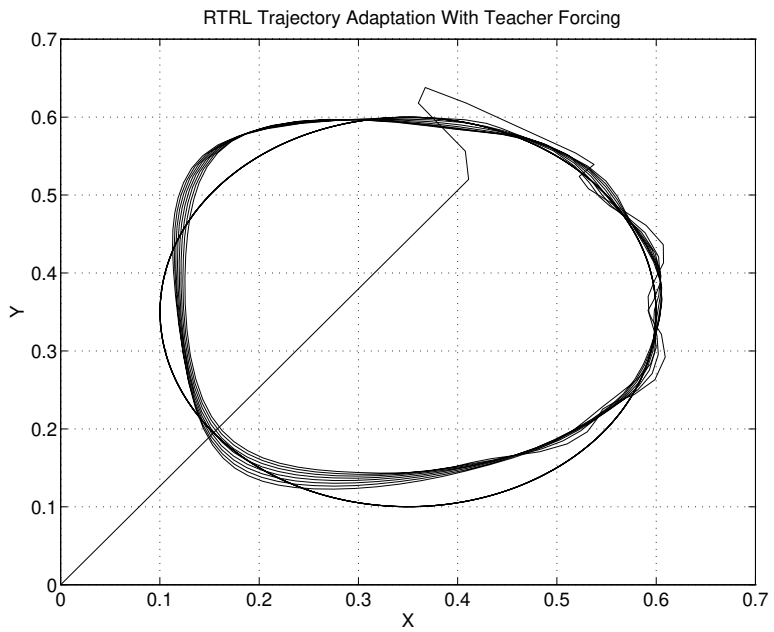


Figure B.3: An X-Y plot of the circle task as was learned by an RTRL algorithm.

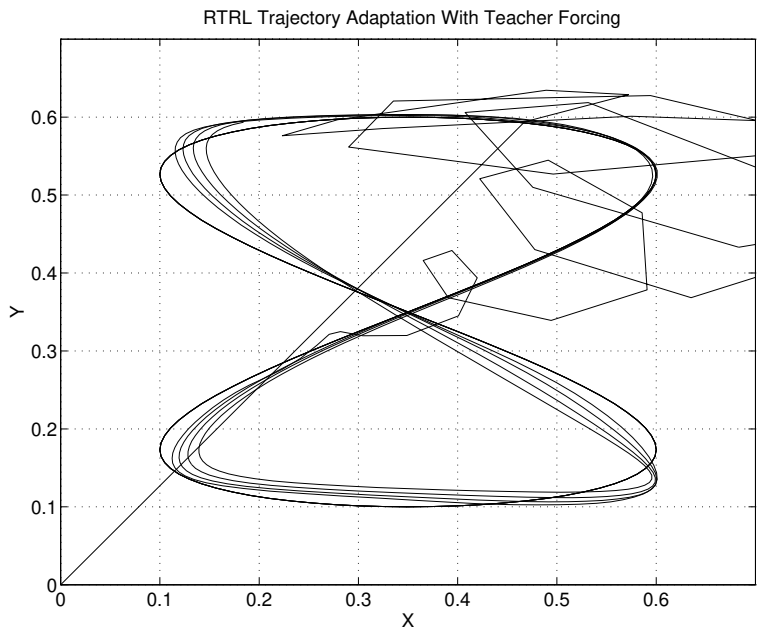


Figure B.4: An X-Y plot of the eight shape task as was learned by an RTRL algorithm.

B.2 State space trajectories in RNN

The work by Pearlmutter on state space trajectory [5] had a great influence on our research: It defined the benchmark problems that are used to test RNN and TDRNN (the "circle" and the "eight figure") and defined the basic TDRNN topology that was used in our work. The dynamic representation for RNN is of the following form (note that Eq. (1.1) and is derived out of the following equations):

$$T_i \dot{y}_i = -y_i + \sigma(x_i) + I_i \quad , \tag{B.5}$$

where

$$x_i = \sum_j w_{ji} y_j \quad . \tag{B.6}$$

The energy (or cost) function is calculated in the usual form :

$$E = \int_t \sum_i (y_i(t) - d_i(t))^2 dt \quad . \tag{B.7}$$

The temporal error property is thus: $e_i(t) = (y_i(t) - d_i(t))^2$. The training is accomplished by defining a back propagating function $z_i(t)$ that is measuring the influence of an infinitesimal change in a

neuron value at time t on the energy function. An equation for z thus has boundary condition at $t = t_f; z(t_f) = 0$ and is calculated by propagating it backwards in time.

$$\dot{z}_i(t) = \frac{1}{T_i} z_i - e_i - \sum_j \frac{1}{T_i} w_{ij} \dot{\sigma}(x_j) z_j \tag{B.8}$$

After obtaining the values of $\mathbf{y}$, $\mathbf{x}$ and $\mathbf{z}$ as a function of time, it is straight forward to compute the parameter correction equations:

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_i} \int_t y_i \dot{\sigma}(x_j) z_j dt \tag{B.9}$$

$$\frac{\partial E}{\partial T_i} = -\frac{1}{T_i} \int_t \dot{y}_i z_i dt \tag{B.10}$$

The results of training the typical training tasks is in table B.2

Task	results
"circle"	6 neurons 12,000 iterations
"eight figure"	12 neurons 20,000 iterations

Table B.2: The performance of indirect GD scheme for RNN

B.3 Adjoint Operators method

This method by Barhen and Toomarian [8] showed a way to reduce the calculation required for training RNN with GD methods.

Task	results
"circle"	6 neurons 300-800 iterations Teacher forcing
"eight figure"	6 neurons 500-1,000 iterations Teacher forcing

Table B.3: The performance of adjoint GD scheme for RNN

Examination of the method, reveals that combining time delays into the network eliminates the adjoint operators advantages and thus making it useless for TDRNN training. The network dynamics

equation is as follows:

$$\dot{y_n} = -k_n y_n + \sigma \left[\gamma_n \sum_m w_{nm} y_m(t) + I_n(t) \right] \tag{B.11}$$

where $\mathbf{y}$ is the neurons output vector, $\mathbf{k}$ is the time constants vector, σ is the neural function, γ is a multiplier vector, $\mathbf{w}$ is the weight matrix and $\mathbf{I}$ is the external input vector. The method is based on stating the adjoint equation of the derivation of equation (B.11) with respect to the whole parameter set (In other words, taking the direct gradient descent approach, and using the adjoint equations). The direct derivation equations are:

$$\frac{\partial \dot{y_n}}{\partial P_l} + \sum A_{nm} \frac{\partial y}{\partial P_l} = \frac{\partial S_n}{\partial P_l} \tag{B.12}$$

where P_l is one of the parameters of the neural network,

$$A_{nm} = \left(k_n - \gamma_n \dot{\sigma}[..] \frac{\partial I_n}{\partial y_n} \right) \delta_{nm} - \gamma_n \dot{\sigma}[..] w_{nm} \tag{B.13}$$

and

$$\frac{\partial S_n}{\partial P_l} = -y_n \delta_{P_l,k} + \gamma_n \dot{\sigma}[..] \sum_m y_m \delta_{P_l,w} + \dot{\sigma} \left(\sum_m w_{nm} y_m + I_n \right) \delta_{P_l,\gamma}. \tag{B.14}$$

And the adjoint equation is as follows:

$$-\dot{v}_n(t) + \sum_m A^T_{nm} v_m = S^*_n \tag{B.15}$$

where the three terms are the adjoint forms of the terms in Eq. (B.13). This adjoint equation reduces the calculation volume if the source term **S** is independent of **y**. This can be achieved without time delays and is shown in [8]. The advantage of this scheme over conventional and indirect GD methods is saving calculations as described in table B.3 (with teacher forcing).

B.4 Signal reconstruction with TDFF

This work by Day and Davenport [10] showed the feasibility of learning to perform complex tasks by a large network. The network configuration they used is Feed Forward network with Timed Delays

(TDFF). This topology was carefully adapted to the type of task that was performed by the network. The task was to reconstruct the Mackey-Glass chaotic time series. The network dynamics is as follows:

$$y_i(t) = \sum_j \sum_k w_{ijk}(t) o_j \left(t - \tau_{ijk}(t)\right) \tag{B.16}$$

where $o_i(t) = \sigma(y_i(t))$. Note that this TDFF configuration consists of multi connections where each connection between neurons can consist of N time delayed connections. That is the reason for the triple index in the equation. the actual network consist of 2 layers of 10 neurons each connected to an input linear neuron and to an output linear network. Training was achieved through a back propagation scheme (using the difference property Δ_j) where in the output layer

$$\Delta_j = 2(d_j(t) - o_j(t))\dot{\sigma}(y_j(t)) \tag{B.17}$$

and for the other neurons

$$\Delta_j = \dot{\sigma}(y_j(t - a_i) \sum_i \sum_k \Delta_i(t - [a_i + \tau_{ijk}]) w_{ijk}(t) \tag{B.18}$$

where index i is referring to the neurons that their output is an input to neuron j and k for the neurons that receive signals from neurons j. The parameter a_i is the maximum delay in the connections getting out of neuron i. The parameter correction is obtained for the weights through

$$\frac{dw_{ijk}}{dt} = \mu o_j(t - [a_i + \tau_{ijk}])\Delta_i(t) \tag{B.19}$$

using the learning rate μ and for the time delays through

$$\frac{d\tau_{ijk}}{dt} = \eta w_{ijk}(t)\dot{\sigma}y_j(t)(t - [a_i + \tau_{ijk}])\Delta_i(t) \tag{B.20}$$

using the learning rate η. The training session was extremely long (acceded 30,000,000 iterations). In fact the time delay element here can be transformed into a regular BP FF network with no time delays. This work emphasized the ability to downsize the network using time delays.

Appendix C

Indirect gradient descent algorithm for TDRNN MODEL 2

Training MODEL2 is significant different in comparison to the original algorithm of the basic model due to its different dynamic behavior. We derive this training algorithm in a similar way to what has been done for the basic model. To compute the function $z(t)$

we first introduce a discrete version of the dynamic equation :

$$y_i(t+\Delta t) = \left(1 - \frac{\Delta t}{T_i}\right) y_i(t) + \frac{\Delta t}{T_i}\left(\sum_j w_{ji}\sigma(y_j(t-\tau_{ji})) + I_i(t)\right) \tag{C.1}$$

In order to compute $z(t)$ we change $y(t)$ with an infinitesimal value ϵ at time t:

$$y_i(t+\Delta t,\epsilon) = \left(1 - \frac{\Delta t}{T_i}\right) (y_i(t)+\epsilon) + \frac{\Delta t}{T_i}\left(\sum_j w_{ji}\sigma(y_j(t-\tau_{ji})) + I_i(t)\right) \tag{C.2}$$

Using the chain rule:

$$\begin{aligned} z_i(t) = \frac{dE(t)}{d\epsilon} = & \underbrace{\sum_j \frac{\partial E}{\partial y_j(t+\Delta t)} \frac{\partial y_j(t+\Delta t)}{\partial \epsilon}}_{I} \\ & + \underbrace{\frac{dE}{d\epsilon}}_{II} + \underbrace{\sum_j \frac{\partial E(t+\Delta t)}{\partial y_j(t+\tau_{ji})} \frac{\partial y_j(t+\tau_{ij})}{\partial \epsilon}}_{III} \end{aligned} \tag{C.3}$$

Where term I is the influence through $y(t)$, term II is the direct influence and term III is the influence through $y(t+\tau_{ji})$. After introducing y into the equation we get the three expressions:

$$
\begin{aligned}
I &= z_j(t+\Delta t)\left(1-\frac{\Delta t}{T_j}\right) \\
II &= e_i \Delta t \\
III &= z_j(t+\tau_{ij})\frac{\Delta t}{T_i}\left(\sum_l w_{li}\dot{\sigma}(y_l(t+\tau_{jl}))\right)
\end{aligned}
\tag{C.4}
$$

The function z is therefore:

$$
\begin{aligned}
z_i(t) = e_i(t)\Delta t + z_i(t+\Delta t)(1+\frac{\Delta t}{T_i}) \\
+\sum_j z_j(t+\tau_{ij})\frac{\Delta t}{T_j}\left(\sum_l w_{li}\dot{\sigma}\left(y_l(t+\tau_{li})\right)+I_i(t)\right)
\end{aligned}
\tag{C.5}
$$

Now we can compute the influence of the parameters on the energy while using the function z. First we compute the influence from the weights w_i while defining a new function V_{ji} so that:

$$
\frac{dE}{dw_{ji}} = \int_{t_0}^{t_f} \frac{dV_{ji}}{dt} dt \tag{C.6}
$$

We define V_{ji} as:

$$
V_{ji} = \frac{dE}{d\epsilon} = \underbrace{\frac{\partial E}{\partial y(t+\Delta t)}}_{=z(t+\Delta t)} + \underbrace{\frac{\partial y(t+\Delta t)}{\partial \epsilon}}_{II} + \underbrace{\frac{dE}{d\epsilon}}_{=0} \tag{C.7}
$$

In order to get term II we define the discrete y function with an infinitesimal change ϵ in the weights w:

$$y_i(t+\Delta t,\epsilon) = \left(1-\frac{\Delta t}{T_i}\right)(y_i(t))+\frac{\Delta t}{T_i}\left(\sum_j (w_{ji}+\epsilon_{ji})\sigma(y_j(t-\tau_{ji}))\right) \tag{C.8}$$

And term II becomes:

$$\frac{\partial y_i(t+\Delta t)}{\partial \epsilon} = \frac{\Delta t}{T_i}\left(\sum_j \dot{\sigma}(y_j(t-\tau_{ji}))\right) \tag{C.9}$$

We get an expression for V_{ji}:

$$\frac{\partial V_{ji}}{\partial t} = -\frac{z_i(t)}{T_i}\left(\sum_j \dot{\sigma}(y_j(t-\tau_{ji}))\right) \tag{C.10}$$

We shift the time $t \rightarrow t+\tau_{ji}$ and get:

$$\frac{\partial V_{ji}}{\partial t} = -\frac{z_i(t+\tau_{ji})}{T_i}\left(\sum_j \dot{\sigma}(y_j(t))\right) \tag{C.11}$$

and get the correction equation where the parameters correction equations are proportional to the derivative expressions:

$$\frac{\partial E}{\partial w_{ji}} = -\int_{t_0+\tau_{ji}}^{t_1+\tau_{ji}} \frac{z_i(t+\tau_{ji})\left(\sum_l \dot{\sigma}(y_l(t))\right)}{T_i} dt \tag{C.12}$$

And now for computing the influence of T on the energy we use an inverse function $\rho_i = \frac{1}{T_i}$. Then we define a new function K_i that complies with:

$$\frac{dE}{d\rho_i} = \int_{t_0}^{t_f} \frac{dK_i}{dt} dt \tag{C.13}$$

We define therefore K_i as:

$$K_i = \frac{dE}{d\epsilon} = \underbrace{\frac{\partial E}{\partial y(t+\Delta t)}}_{=z(t+\Delta t)} + \underbrace{\frac{\partial y(t+\Delta t)}{\partial \epsilon}}_{II} + \underbrace{\frac{dE}{d\epsilon}}_{=0} \tag{C.14}$$

In order to get term II we define the discrete y function with an infinitesimal change ϵ in the parameter ρ:

$$y_i(t+\Delta t, \epsilon) = (1 - \Delta t(\rho_i + \epsilon)(y_i(t)) + (\rho_i + \epsilon)\Delta t(x_i) \tag{C.15}$$

We also note that

$$\frac{dy_i}{d\epsilon} = -\Delta t \; y_i + \Delta t(x_i) = \frac{dy_i}{dt} \Delta t \; T_i \tag{C.16}$$

Then,

$$\Delta K_i = z_i(t+\Delta t)\frac{dy_i}{dt}\Delta t T_i \tag{C.17}$$

and

$$\frac{dK_i}{dt} = z_i \dot{y}_i T_i \tag{C.18}$$

where $\dot{y} = \frac{dy}{dt}$. To translate the influence of ρ to the influence of T we use the chain rule again:

$$\frac{dE}{dT_i} = -\frac{dE}{d\rho_i}\frac{1}{T_i^2} = \frac{-z_i \dot{y}_i}{T_i} \tag{C.19}$$

We finally get

$$\frac{dE}{dT_i} = -\int_t \frac{z_i \frac{dy(t)}{dt}}{T_i} dt \tag{C.20}$$

And the influence of time delays τ_{ji} on E is computed by defining a new function H where

$$\frac{dE}{d\tau_{ji}} = \int_{\tilde{t}} \frac{dH_{ji}}{dt} dt \tag{C.21}$$

We define therefore H_{ji} as:

$$H_{ji} = \frac{dE}{d\epsilon} = \underbrace{\frac{\partial E}{\partial y(t+\Delta t)}}_{=0} \underbrace{\frac{\partial y(t+\tau_{ji})}{\partial \epsilon}\frac{\partial E(t+\tau_{ji})}{\partial y(t+\tau_{ji})}}_{II} + \underbrace{\frac{dE}{d\epsilon}}_{=0} \tag{C.22}$$

In order to get term II we define the discrete y function with an infinitesimal change ϵ in the delays τ:

$$y_i(t+\Delta t,\epsilon) = \left(1-\frac{\Delta t}{T_i}\right)(y_i(t)) + \frac{\Delta t}{T_i}\left(\sum_j w_{ji}\sigma(y_j(t-\tau_{ji}-\epsilon))\right) \tag{C.23}$$

Expression II is then:

$$\frac{\partial y(t+\tau_{ji})}{\partial \epsilon} = \frac{\Delta t}{T_i}\left(\sum_j w_{ji}\dot{\sigma}(y_j(t))\right) w_{ji}\dot{y}_j(t) \tag{C.24}$$

We get an expression for H_{ji}:

$$\Delta H_{ji} = -z_i(t+\tau_{ji})(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)\frac{\Delta t}{T_i} \tag{C.25}$$

$$\frac{\partial H_{ji}}{\partial t} = -z_i(t+\tau_{ji})(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)\frac{1}{T_i} \tag{C.26}$$

and get the correction equation where the parameters correction equations are proportional (with a negative sign) to the derivative expressions:

$$\frac{\partial E}{\partial \tau_{ji}} = -\frac{1}{T_i}\int_{t_0+\tau}^{t_1+\tau} z_i(t+\tau_{ji})(x_i(t+\tau_{ji}))w_{ji}\dot{y}_j(t)dt \tag{C.27}$$